Towards A Better Understanding of Browser Fingerprinting

Submitted by

Nasser Mohammed Al-Fannah

for the degree of Doctor of Philosophy

of the

Royal Holloway, University of London

2019

Declaration

I, Nasser Mohammed Al-Fannah, hereby declare that this thesis and the work presented in it is entirely my own. Where I have consulted the work of others, this is always clearly stated.

Signed.....(Nasser Mohammed Al-Fannah)
Date:

To my family ♡

Abstract

Browser fingerprinting is a relatively new method of uniquely identifying browsers that can be used to track web users. Collectively unique and hence identifying pieces of information, making up what is known as a fingerprint, can be collected from browsers by a visited website, e.g. using JavaScript. Browser fingerprinting is increasingly being used for online tracking of users even in the absence of a persistent IP address or cookie. Since this represents a major threat to user privacy, it is therefore extremely important to understand better how it works, how widely it is being used, and how its use can be controlled. This observation motivates the work described in this thesis.

We automatically crawled the most visited 10,000 websites; this gave insights into the number of websites that are potentially using fingerprinting, which websites are collecting fingerprinting information, and exactly what information is being retrieved. We found that approximately 69% of websites are, potentially, involved in first-party or third-party browser fingerprinting.

We examined the fingerprintable attributes made available by a range of modern browsers. We tested the most widely used browsers for both desktop and mobile platforms. The results reveal significant differences between browsers in terms of their fingerprinting potential, meaning that the choice of browser has significant privacy implications.

The recently-introduced WebRTC API enables client IP addresses to become available to a visited website via JavaScript, even if a VPN is in use. We performed experiments with the five most widely used WebRTC-enabled browsers and five widely used commercial VPN services to investigate this further. Our experiments revealed that the number and type of leaked IP addresses are affected by the choice of browser as well as the VPN service and program settings.

We describe *FingerprintAlert*, a freely available browser extension we developed that detects and, optionally, blocks fingerprinting attempts by visited websites. It also provides users with a log of detected fingerprinting attempts including what information was collected and by whom.

Our investigations confirm previous findings that third-party fingerprinting countermeasures have inherit limitations and many browser vendors do not appear to have made significant efforts to control browser fingerprinting. To help provide a better understanding of the problem, we provide a comprehensive and structured discussion of measures to limit or control browser fingerprinting, covering both user-based and browser-based techniques. Further, a somewhat counterintuitive possible new browser identifier is proposed which could make cookies and fingerprint-based tracking redundant; the need for, and possible effect of, this feature is discussed.

Acknowledgements

First and foremost I thank God for the countless blessings he bestowed upon me.

I am thankful beyond words to my supervisor, and *mentor*, Professor Chris Mitchell who has patiently guided and supported me during the past four years.

I would like to thank my colleagues and friends, who I have been fortunate to work with in of our reading group, Zhaoyi Fan, Angela Heeler, Mohammed Shafiul Alam Khan, Wanpeng Li, Gaëtan Pradel, Zhang Xiao, and my wife Fatma Al-Maqbali. They have truly created a very friendly atmosphere in which we exchanged ideas, feedback,... and laughs.

I extend my thanks to my sponsor, the Ministry of Higher Education and the College of Applied Sciences, Sultanate of Oman.

I would also like to give a very special thanks to my mother Suad and my father Mohammed. They both have *always* been there for me to support, teach and encourage throughout the years.

Last, but certainly not least, I thank my wife Fatma and my daughter Azzahraa as they have been, without a doubt, my driving force all these years and are still. They are truly one of the greatest blessings in my life.

Contents

List of Figures

List of Tables

List of Abbreviations

API	**A**pplication **P**rogramming **I**nterface
CSS	**C**ascading **S**tyle **S**heets
FI	**F**ingerprintability **I**ndex
GPU	**G**raphics **P**rocessing **U**nit
HDD	**H**ard **D**isk **D**rive
HTML	**H**yper**t**ext **M**arkup **L**anguage
HTTP	**H**yper**t**ext **T**ransfer **P**rotocol
HTTPS	**H**yper**t**ext **T**ransfer **P**rotocol **S**ecure
ICE	**I**nteractive **C**onnectivity **E**stablishment
ID	**Id**entifier
IP	**I**nternet **P**rotocol
IPsec	**I**nternet **P**rotocol **Sec**urity
IPv4	**I**nternet **P**rotocol **v**ersion **4**
IPv6	**I**nternet **P**rotocol **v**ersion **6**
L2TP	**L**ayer **2** **T**unneling **P**rotocol
LAN	**L**ocal **A**rea **N**etwork
NAT	**N**etwork **A**ddress **T**ranslator
ORTC	**O**bject **R**eal-**T**ime **C**ommunications
OS	**O**perating **S**ystem
PC	**P**ersonal **C**omputer
PPTP	**P**oint-to-**P**oint **T**unneling **P**rotocol
SSD	**S**olid-**S**tate **D**rive
SSL	**S**ecure **S**ockets **L**ayer
STUN	**S**ession **T**raversal **U**tilities for **N**AT
TCP	**T**ransmission **C**ontrol **P**rotocol
TLS	**T**ransport **L**ayer **S**ecurity
TURN	**T**raversal **U**sing **R**elay **N**AT
UDP	**U**ser **D**atagram **P**rotocol
ULA	**U**nique **L**ocal **A**ddress
URL	**U**niform **R**esource **L**ocator
VPN	**V**irtual **P**rivate **N**etwork
WebGL	**Web** **G**raphics **L**ibrary
WebRTC	**Web** **R**eal-**T**ime **C**ommunication

Chapter 1

Introduction

1.1 Context of Research

A number of authors have discussed the very wide variety of readily available attributes collectable by websites from a visiting browser. Because the set of retrievable attributes is in most cases unique per browser instance, this enables websites to uniquely identify browsers; this is known as *browser fingerprinting* [1, 18, 52, 61]. The main reason browser fingerprinting has attracted so much attention is that it enables tracking of user platforms, i.e. linking of multiple visits by a single browser instance to the same or multiple websites.

Because of its potential use for tracking, browser fingerprinting is becoming an increasingly serious privacy concern despite some apparently benign applications. Its virtually permanent nature is something that might be subject to future regulation, much as the use of cookies has recently received the attention of regulators in Europe [21]. Several studies (e.g. [47, 57, 86]) have suggested that browser fingerprinting has been used to track users by re-spawning tracking cookies and, as a result, limit the degree to which user cookie deletion protects user privacy. Fingerprinting is also potentially being used as an alternative to cookie tracking. It is interesting to note that the use of browser fingerprinting by websites is to some extent regulated by the European Union's General Data Protection Regulation (GDPR) [14] that was adopted in 2016[1]. However, browser fingerprinting remains widely used and appear to be largely uncontrolled.

Browser fingerprinting use is virtually invisible to users and there is no direct way of preventing it. Moreover, we found that the four browsers used by more than 88% of web users (i.e. Chrome, Internet Explorer, Firefox and Edge) do almost nothing to help

[1] Given that browser fingerprinting involves collecting user-related data as well as potentially tracking users, GDPR would appear to require websites to both announce and justify its use [78].

mitigate fingerprinting[2], alert the user to its occurrence, or even provide information about it in user help documents.

This chapter provides an overview of the thesis, and is organized as follows. In Section 1.2 we discuss the motivation for the research described in this thesis. Section 1.3 outlines the major contributions. The role of co-authors in the work described in this thesis is clarified in Section 1.4. A list of relevant ublications is presented in Section 1.5, and Section 1.6 concludes the chapter with a brief outline of the thesis.

1.2 Motivation

The thesis attempts to answer the following question. *To what extent does real world browser fingerprinting affect user privacy?*

As noted above, in recent years browser fingerprinting has been widely discussed not least because of its potential use for user tracking. This is a fast-moving area, and hence findings from only a few years ago may no longer give a true picture of its current effectiveness and prevalence. In particular, modern browsers are constantly adding new features with the introduction of new APIs such as Canvas, WebGL, and WebRTC. These new additions have an unintended effect of increasing the fingerprinting surface of browsers [17, 66]. Indeed, the range of attributes retrievable from browsers, as well as methods for retrieving them, have been widely discussed. However, relatively little has been published regarding the real-world prevalence of browser fingerprinting, who is deploying it, the types of attributes collected to achieve it and how to control it. In addition, no previous author has compared how browsers differ in terms of their susceptibility to fingerprinting.

Because of the serious privacy concerns arising from fingerprinting-enabled tracking, these issues clearly merit further investigation, and this observation has motivated the work described in this thesis.

1.3 Contributions

In this thesis, we performed a number of studies to learn more about the nature and prevalence of browser fingerprinting. In the first of these we attempted to gain a better understanding of the degree to which widely visited websites are performing or enabling browser fingerprinting.

- We assembled a script that crawled the 10,000 most widely visited websites to detect the unprompted collection of information that could be used for fingerprint-

[2]Firefox has a limited set of options to reduce the effectiveness of fingerprinting.

ing. We were able to identify websites that collect such information as well as the types of information they collected. Whilst we are not the first to perform such a study, we used a novel method of identifying possible browser fingerprinting that provides an upper bound on the number of sites that perform it.

In our second group of contributions, we examine specific aspects of fingerprinting techniques, with the goal of understanding their effectiveness. As part of this work, we were the first to demonstrate that browsers vary in their susceptibility to browser fingerprinting. In related work, we also examined one of the techniques that could be used to compromise user privacy through the exploitation of a feature in the WebRTC API. This API can be used to learn a client's private IP address(es) even if the client is connected via an anonymizing VPN. We examined the extent to which this privacy vulnerability can be exploited in widely used browsers and VPN clients.

- The most widely used browsers were tested and shown to vary in the amount of information useful for fingerprinting that can be retrieved. This means that some browsers are more likely to enable an instance to be uniquely identified.

- We conducted a study to determine whether a recently proposed means of learning private client IP addresses via the WebRTC API works in practice; in particular, we focussed on the case where the client is using a VPN, often with the specific goal of enhancing privacy. In this study we examined every combination of five widely used browsers and five widely used VPN clients, in each case enumerating which IP addresses could be obtained.

Whilst the first chapters of the thesis are directed at understanding the prevalence and effectiveness of browser fingerprinting methods, the rest of the thesis is aimed at trying to improve the current situation, by giving users greater control over fingerprinting. We describe the design and implementation of a browser extension that alerts users to browser fingerprinting attempts, and gives them the option to block them. Furthermore, we analyze all previously discussed fingerprinting countermeasures, and use this analysis to formulate a set of recommendations that could help control browser fingerprinting. Finally, we propose a novel type of browser identifier with the goal of obviating the need for browser fingerprinting, which could help bring to an end its current widespread and uncontrolled use.

- A detailed description is given of a novel browser extension. The extension helps users be aware of fingerprinting attempts by websites they visit. Moreover, it can also help to reduce the effectiveness of browser fingerprinting.

- We provide comprehensive guidance on how browser fingerprinting can be controlled, especially by browser vendors. We also propose a new type of browser identifier that could provide a standardized alternative to browser fingerprinting. The identifiers would enable online tracking but, unlike fingerprinting, would be transparent and user controllable.

1.4 Joint Work

With the following exceptions, all relating to software development, I have performed all the research described in this thesis, under the supervision of Professor Chris Mitchell.

- The coding for the crawler software used to perform the experiments described in Chapter 3 was mainly performed by Dr Wanpeng Li, although decisions about functionality were primarily under my control.
- The scripts that support the Fingerprintability website, discussed in Chapters 4 and 5, were jointly developed with Dr Wanpeng Li.
- Coding for the FingerprintAlert browser extension, described in Chapter 6, was mainly performed by Dr Wanpeng Li, although user interface design and decisions about functionality were primarily under my control.

1.5 Publications

Publications containing some of the research results described in this thesis are listed below.

- N. M. Al-Fannah and W. Li. Not all browsers are created equal: Comparing web browser fingerprintability. In S. Obana and K. Chida, editors, *Advances in Information and Computer Security — 12th International Workshop on Security, IWSEC 2017, Hiroshima, Japan, August 30 – September 1, 2017, Proceedings*, volume 10418 of *Lecture Notes in Computer Science*, pages 105–120. Springer, 2017. [**Winner of best student paper**.]
- N. M. Al-Fannah. One leak will sink a ship: WebRTC IP address leaks. In *International Carnahan Conference on Security Technology, ICCST 2017, Madrid, Spain, October 23–26, 2017*, pages 1–5. IEEE, 2017.
- N. M. Al-Fannah, W. Li, and C. J. Mitchell. Beyond cookie monster amnesia: Real world persistent online tracking. In L. Chen, M. Manulis, and S. Schneider, editors,

Information Security — 21st International Conference, ISC 2018, Guildford, UK, September 9–12, 2018, Proceedings, volume 11060 of *Lecture Notes in Computer Science*, pages 481–501. Springer, 2018.

1.6 Thesis Outline

The remainder of this thesis is organized as follows.

- **Chapter 2** gives the background material necessary for the remainder of the thesis. It describes the various aspects of the browser fingerprinting ecosystem that are necessary to understand the rest of the thesis.
- **Chapter 3** is concerned with gaining a better understanding of the current prevalence of browser fingerprinting as well as the extent of its tracking capabilities. It first examines previous work aimed at determining the prevalence of browser fingerprinting in the Web. Then it describes a study we conducted to identify the possible conduct of browser fingerprinting by the top 10,000 websites.
- **Chapter 4** reviews the differences between widely used browsers in terms of their susceptibility to browser fingerprinting. The tests were performed on a range of browsers and operating systems (both desktop and mobile).
- **Chapter 5** describes the results of a study on the leakage of client IP address(es) even when using an anonymizing VPN. This leak of IP addresses can be very useful for browser fingerprinting, and is also a major privacy threat in that in some cases it can reveal a user's geographic locality. We examined the presence of this problem in a range of widely used VPN clients and browsers.
- **Chapter 6** describes a novel anti-fingerprinting browser extension, *FingerprintAlert*. It details the operation of the *FingerprintAlert* extension which was used in the study described in Chapter 3 and is also of value in its own right.
- **Chapter 7** considers the degree to which browser fingerprinting can be controlled by examining all previously proposed countermeasures, and assessing their real-world effectiveness. This leads to a set of recommendations for browser fingerprinting control, as well as a proposal for the use of a standardized browser identifier to help make browser fingerprinting unnecessary.
- **Chapter 8** concludes the thesis by summarizing the main contributions as well as highlighting possible areas for future work.

Chapter 2

Background

2.1 Introduction

In this chapter we provide background information that is necessary to understand the rest of thesis. We are concerned throughout this thesis with HTTP interactions between a browser running on a user platform, e.g. a phone or PC (the client), and a remote (web) server. Browser fingerprinting relies on the server gaining information about the client platform via HTTP, including via executable code (typically in JavaScript) downloaded to the client as part of an HTTP exchange and subsequently executed on that client. Downloaded JavaScript code is typically able to access a wide range of internal browser APIs[1] completely transparently to the user, and these APIs can reveal a range of information about the client which can be reported back to the web server, again completely transparently to the user.

Client-server interactions supporting browser fingerprinting may also include the involvement of third-party web servers, to which the client will send HTTP requests as a result of links included in the HTTP message sent to the client. We do not go into the technical details of HTTP and JavaScript here — the interested reader is referred to one of the many textbooks on the subject, e.g. Crockford [15] or Gorley and Totty [30], as well as RFC 7231 [25].

The remainder of the chapter is organized as follows. Section 2.2 provides an overview of browser fingerprinting. Some key techniques commonly used for browser fingerprinting are described in Section 2.3. Section 2.4 provides an overview of online tracking. We discuss privacy concerns related to browser fingerprinting in Section 2.5. The summary in Section 2.6 concludes the chapter.

[1] A web API, i.e. a web Application Programming Interface, is a set of functions or methods that can be used to access certain functionality of web browsers.

2.2 Browser Fingerprinting

2.2.1 Fundamentals

Browser fingerprinting is a technique that can be used by a web server to uniquely identify a platform; it involves using information provided by the browser, e.g. website-originated JavaScript, to assemble a platform-specific *fingerprint*. The differences in web browser instances were first discussed 10 years ago by Mayer [51], albeit in a very limited way. A year later, Mayer's work was followed by the seminal work of Eckersley [18]. Eckersley coined the term *browser fingerprinting* and performed a detailed study, including a number of important experiments. Since then, the range and richness of fingerprinting information retrievable from a browser has substantially increased [44]. Of course, cookies[2] and/or the client IP address can also be used for platform identification, but browser fingerprinting is designed to enable identification even if cookies are not available and the IP address is obfuscated, e.g. through the use of anonymizing proxies.

Back in 2010, Eckersley [18] described how the collection of a range of apparently trivial and readily-available browser attributes, such as time zone, screen resolution, set of installed plugins, and operating system version, could be combined to uniquely identify a browser instance; this is the process he dubbed browser fingerprinting. Since then, many other authors, including Mowery and Shacham [56], Mowery et al. [55], Boda et al. [10], Olejnik et al. [63], Fifield and Egelman [26] and Mulazzani et al. [57], have described a range of ways of enhancing its effectiveness. In parallel, and motivated by the threat to user privacy posed by browser fingerprinting, a number of authors, e.g. Nikiforakis et al. [60], Fiore et al. [27] and FaizKhademi et al. [22] have proposed ways of limiting its effectiveness.

The BrowserLeaks website[3] (`https://www.browserleaks.com`) and Alaca and Van Oorschot [6] catalogue a wide range of types of information that could be used for browser fingerprinting. Pathilake et al. [83] have also classified some of the most widely used methods for fingerprinting. As discussed in 2.2.2, browser fingerprinting is clearly very effective.

Browser fingerprinting enables user web activity to be tracked. It relies on learning properties of a browser and its host platform, including both hardware properties and software state (cf. the term *device fingerprinting* [28]). Browser fingerprinting typically

[2] A web cookie is a small amount of data sent by a website as part of an HTTP response and then stored by the browser. The browser then provides the contents of the cookie back to the same server in subsequent HTTP requests [7].

[3] This website is the most comprehensive source for fingerprintable attributes we were able to find on the web. It uses a variety of scripts to display retrievable fingerprintable attributes. Most fingerprintable attributes discussed in the literature can be found on the website as well as many that are not.

involves a web server performing some combination of: (a) collecting and analyzing information contained in HTTP request headers, and (b) downloading JavaScript to the browser which collects and sends back information gathered from browser APIs. Examples of collected information include: screen resolution, CPU/GPU model, and names of installed fonts[4]. As in these examples, collectable attributes relate to both browser and host platform. Some attributes change over time (e.g. browser version) but uniquely identifying a browser instance is usually still possible [86], and uniquely identifying the hosting platform may still be possible even if a different browser is used [12].

2.2.2 Effectiveness

Browser fingerprinting is clearly very effective; for example, in a large-scale study, Laperdrix et al. [44] observed that an average of 86% of desktop and mobile browsers possess a unique fingerprint; other studies [18, 55] have reported similar results (80–90%). It is important to note that some of the attributes that can be used for fingerprinting vary between desktop and mobile platforms; as a result the efficiency of fingerprinting also varies between platform types [44]. For example, a device model name can be retrieved from a mobile browser's user agent HTTP header field but not from its desktop counterpart. This is an issue examined in detail in Chapter 4.

2.2.3 Applications

Apart from client tracking, discussed in Section 2.4, four widely discussed uses of browser fingerprinting are: targeted advertising [2, 45]; social media sharing [45, 68]; analytics services [2, 45]; and web security [2, 82]. Browser fingerprinting also has other potential uses, e.g. to act as a second layer of authentication [18]. However, even in these cases the server gets the benefit, and the user is often not informed that fingerprinting is in use [90]. Determining the exact reason(s) why a website deploys browser fingerprinting is extremely difficult.

2.2.4 Third-Party Fingerprinting

Browser fingerprinting websites perform it either as a first-party or a third-party (or both). That is, a website may download a piece JavaScript code to the browser, which can send the collected attributes back to either its own site (first-party fingerprinting) or to a third-party site (third-party fingerprinting) [70]. It is even possible that some

[4] A demonstration of the wide range of information collectable from any browser is available at https://fingerprintable.org/test.

website operators are not aware that a third-party is performing browser fingerprinting via their website [20]. This could arise because third-party fingerprinting sites typically provide client websites with the JavaScript code which collects and sends back to the third-party site the attributes used for fingerprinting; in return for serving this JavaScript, the third-party site provides a range of services to the client website (e.g. data analytics or social plugins). As a result, some website operators may not know what data the third-party JavaScript script collects from user browsers, or what it might be used for.

In the context of tracking, first-party fingerprinting gives relatively little information to a website — it merely enables multiple visits by the same browser to be linked, and gives no information about other visited websites. If the user identity is known by other means (e.g. because the user logs in) it can also indicate when this user is employing multiple devices [2]. Third-party fingerprinting, on the other hand, is much more privacy-damaging in that it enables browsers (and hence users) to be tracked across multiple websites. In Chapter 3 we report on the websites that perform the majority of third-party tracking.

2.3 Browser Fingerprinting Techniques

As first noted by Eckersley [18], one fundamental source of client information that can be used to help fingerprinting is the user agent field of an HTTP request header [25]. It gives information related to the browser, including its type and version. This field can be used to enable a website to tailor content to meet the needs of differing browsing platforms [25]. An example of the contents of this field as generated by the Mozilla Firefox browser running on a Windows 10 PC is given in Figure 2.1. The various components of the example user agent[5] are explained in Table 2.1.

```
Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:66.0) Gecko/20100101
Firefox/66.0.
```

Figure 2.1: A sample user agent string

More generally, browser fingerprinting can be performed by either active or passive means [17]. *Passive* fingerprinting depends entirely on information retrievable through regular HTTP requests, such as the user agent field discussed above, whereas *active* fingerprinting involves the use of scripts to retrieve further information about the browser and its configuration. Active fingerprinting enables websites to collect far

[5]The components are explained following to Mozilla developer's website https://developer.mozilla.org/docs/Web/HTTP/Headers/User-Agent/Firefox [accessed on 16/05/2019]

Table 2.1: Explanation of the various components of a sample user agent string

Component	Meaning
`Mozilla/5.0`	Compatible with the last version of the legacy Mozilla browser*
`Windows NT 10.0`	Operating system is Windows 10
`Win64`	64-bit browser running on Windows
`x64`	64-bit operating system
`rv:66.0`	Rendering engine version
`Gecko`	Rendering engine
`20100101`	Build date 01/01/2010*
`Firefox/66.0`	Browser name and version

*Obsolete fields.

more information than is collectable via passive fingerprinting, and thus makes browser fingerprinting much more effective [6]. However, while active fingerprinting is in principle detectable through analysis of downloaded JavaScript, information useable for passive fingerprinting is automatically received by websites (i.e. without prompting) and that makes it virtually undetectable.

We next provide more details about the types of information that can be used for both passive and active fingerprinting.

2.3.1 Passive Fingerprinting

As noted above, passive fingerprinting uses attributes that are part of routine communications between a client (i.e. browser) and a server (i.e. website). In particular it does not involve any active probing of the client's browser such as the use of JavaScript. Examples of attributes that can be gathered in this passive way include the following.

- **Clock skew** involves identifying the deviation of a client's clock, in milliseconds, from the true time [40]. Clock skew for a client device can be estimated by gathering and processing a number of TCP timestamps.

- As discussed above, the **user agent** [25] HTTP header string provides web servers with information related to the client's browser and hosting platform. This string is sent by a client browser to the web server of a visited website as part of a client HTTP request for web resources. As has been widely discussed [18, 29, 88], this string is a rich source of discriminating information for fingerprinting. According to RFC 7231 [25], the user agent should be included in all HTTP request headers but it is not mandatory. A browser can be configured by a user to exclude the user agent string from its HTTP request headers. The user agent string can also be retrieved if a website used JavaScript to probe the browser. Hence, the user

agent can also be part of active fingerprinting (see 2.3.2).

- The **IP address** of a client can be used for browser fingerprinting [18]. Typically, a client device possess at least two IP addresses, one public and another private/local. However, only the *public* IP address is readily known by visited websites. The usefulness of an IP address for fingerprinting varies depending on the client network setup. For example, a dedicated static client IP address is significantly more distinguishing than a dynamically assigned client IP address that is potentially shared by other clients. IP addresses can also be retrieved via active fingerprinting using the WebRTC API (see 2.3.2).

2.3.2 Active Fingerprinting

This involves the execution of attribute-gathering scripts within the client browser. The number of possible attributes of this type that have been discussed in the literature is far larger than the number of passive fingerprinting attributes. Some key examples of attributes of this type are as follows.

- **Canvas fingerprinting** is a widely discussed and highly effective browser fingerprinting technique [56]. The Canvas API is a recently widely-deployed HTML5 API that allows websites to render an image for display by the user browser, a bandwidth-efficient alternative to downloading an image file from the server [87]. A number of authors [1, 44, 56] have demonstrated the possibility of uniquely fingerprinting browsers and their host platforms based on subtle differences in how an image is rendered by the browser.

  This method works because the Canvas API allows a visited website to request the return of the RGBA[6] values of a canvas-rendered image pixels[7] [56]. The details of this image will vary depending on the platform and the browser, meaning that the details of the rendered image can be used as a fingerprinting attribute. For convenience, the image data set can be input to a hash function and the output of this used as the Canvas attribute. This technique is known a *canvas fingerprinting*.

- **WebRTC** [9] is a set of APIs and communications protocols that provides browsers and mobile applications with Real-Time Communications (RTC) capabilities. This set of APIs can be used in a variety of ways to obtain information useful for browser fingerprinting.

[6] **R**ed, **G**reen, **B**lue and **A**lpha (i.e. opacity).
[7] The `canvas.toDataURL()` function is used to achieve this.

For example, identifying one or more of the client IP addresses via a feature of WebRTC was first reported and demonstrated by Roesler[8] in 2015. The disclosure of client private IP address(es) in this way is informally known as a *WebRTC Leak*. The IP address(es) revealed through a WebRTC leak could include the client's local IPv4 address as well as one or more of the client's unique local addresses (ULAs)[9]. As discussed by a number of authors (e.g. Hosoi et al. [35], Jakobsson [37] and Sandholm et al. [72]), a WebRTC-enabled browser uses the Interactive Connectivity Establishment (ICE) [38] Protocol to establish a P2P connection. This protocol enables it to retrieve both public and local IP addresses to establish a connection. To acquire the IP addresses, the ICE protocol utilizes a STUN (Session Traversal Utilities for NAT) [71] server, and if that fails, it relays the connection through a TURN (Traversal Using Relay NAT) [50] server. A tracking website could use STUN/TURN servers to trick a browser into revealing IP addresses that are otherwise invisible to websites.

A different means of using WebRTC for browser fingerprinting arises from the fact that apparently unique IDs are assigned to certain types of hardware device within a client platform. These IDs vary in length and persistence depending on the browser (see Chapter 4) and can potentially be used for fingerprinting since they are made available to executing JavaScript by the WebRTC API.

- The set of **installed fonts** on a device can be used as a fingerprinting attribute, as this set appears to vary widely across devices, [2, 18]. Using this information for browser fingerprinting was first described by Eckersley [18] in 2010, who found it to be one of the most useful techniques from amongst those he examined. The method originally discussed for retrieving this information was through probing a Flash-enabled browser for the list of installed fonts. This method for learning the set of installed fonts is more effective for browser fingerprinting purposes than other techniques because Flash provides the font list sorted in the installation order, where this ordering is itself of value for fingerprinting. The enumeration of fonts through Flash is typically achieved by including a Flash object on a website that allows the retrieval of font information. However, since 2010, this method of retrieving font information is no longer very effective, since Flash is no longer widely used (indeed, many browsers do not allow its use). However, despite this, the set of installed fonts is still a very valuable fingerprinting attribute as this

[8] The report and the demonstration script can be found at https://github.com/diafygi/webrtc-ips [accessed on 24/05/2017]

[9] The ULA is the approximate IPv6 counterpart of the IPv4 local address; see https://tools.ietf.org/html/rfc4193 [accessed on 03/03/2017]

information can be obtained in a number of other ways. For example, the font list can be obtained using JavaScript and Cascading Style Sheets (CSS)[10], as well as via the Canvas API as reported by Englehardt and Narayanan [20].

- The **WebGL** API [36] is used to render graphics within a browser. As part of its functionality it reveals various host attributes (e.g. render buffer size and GPU/CPU model) of the hardware that hosts the browser [6, 44, 56]. In Appendix C.1 we provide a list of WebGL attributes that were collected by websites that were surveyed in the experiment described in Chapter 3.

  Some browsers provide access to the identity of the vendor and the specific model of the user platform's CPU or GPU. These two pieces of information are obtained by requesting the following WebGL attributes: `UNMASKED VENDOR WEBGL` and `UNMASKED RENDERER WEBGL`. These attributes could reveal the Central Processing Unit (CPU) type if there is no GPU or if the GPU is not used by the browser.

- Although the **user agent** is typically received by a web server at first point of contact, it can also be actively retrieved via the JavaScript `navigator.userAgent` property. This potentially allows any third-party script to retrieve a browser's user agent string even though it does not receive the initial HTTP request header from the browser (it is only received by the first-party web server [25]). Moreover, as reported by Nikiforakis et al. [61], some privacy browser extensions spoof the user agent string in the HTTP request header but not in the browser itself and so actively probing a browser for the user agent could reveal the real string.

2.4 Online Tracking

Online tracking (or web tracking) is the process of monitoring a user's online activities; entities that perform tracking are known as *trackers* [45]. It is virtually impossible to determine if a website is actually tracking users; one can simply observe whether a website collects attributes from browsers that would allow it to track a client via browser fingerprinting. In line with common usage, we refer to recipients of fingerprintable data (whether first- or third-party) as trackers or fingerprinters.

In practice, the most common motive for online tracking is to enable online behavioural advertising. This describes the practice by web advertising companies of tracking users' online activities in order to display personalised and targeted advertisements [84]. Additionally, tracking is used as a tool for market research [52]. There are

[10] As reported by Takei et al. [79], installed fonts can be detected through CSS even when JavaScript is disabled on a client's browser.

two main approaches to online tracking — stateful tracking involving the use of cookies [52], and stateless tracking, including the use of browser fingerprinting as defined in Section 2.2. In this thesis, we focus on the latter.

Tracking web users has long been possible by using cookies. However, the absence of a cookie (e.g. because it has been deleted by the user) means that the device is unlikely to continue to be tracked [18]. By contrast, browser fingerprinting requires no files to be stored on the user's device, its effectiveness partly depends on the browser, and users have virtually no control over it. It can be used for tracking web users by creating a unique ID derived by combining collected attributes [44].

That is, browser fingerprinting is in some ways a more reliable method of tracking than the use of cookies [47], and it appears that browser fingerprinting is increasingly being used for this purpose. Unlike browser fingerprinting, cookies are stored on user devices and so can be controlled or deleted by users. In particular, the use of a *private browsing mode*[11], as provided by many browsers, whilst limiting the use of cookies does very little to protect users against browser fingerprinting (as discussed in Chapter 4). Furthermore, while modern browsers provide a user-selectable *Do Not Track* option, this apparently does not prevent widespread tracking [2].

2.5 Browser Fingerprinting Privacy Concerns

As noted above, concern has been expressed about the threat to user privacy posed by browser fingerprinting. Since Eckersley first described it, the range and richness of information retrievable from a browser that is usable for fingerprinting has substantially increased, as has real-world deployment of fingerprinting by websites (see Chapter 3).

As has been widely discussed, for example by Eckersley [18], Narayanan and Reisman [58], and Perry [65], there are a number of reasons why browser fingerprinting represents a more significant threat to user privacy than cookies.

- Typically there is no simple way to determine for certain whether a website is deploying any of the various browser fingerprinting techniques.
- A user can limit the tracking power of cookies in a number of ways, e.g. by regularly deleting cookies or blocking them altogether (as supported by most browsers), but there are no comparable, easily configured, means of limiting fingerprinting.
- Unlike cookies, browser fingerprinting is not dependent on a single explicit feature of HTTP. Fingerprinting rather relies on many techniques to collect various

[11] Modes of this type, which have various names, are intended to enhance the privacy properties of the browser [89].

information about the properties and configuration of the browser and its host platform. Any of this information has the potential to be used for fingerprinting.

2.6 Summary

Browser fingerprinting is a relatively new method of uniquely identifying browser instances that can be used to track web users. A range of individual pieces of information (attributes) about a platform can be collected from a browser by a visited website, e.g. using downloaded JavaScript; whilst typically not individually unique, the set of attributes make up what is known as a fingerprint, and this fingerprint can typically be used to uniquely identify a platform. In some ways browser fingerprinting is more privacy-threatening than tracking via cookies, as users have no direct control over it. It is increasingly being used for online tracking of users even in the absence of a persistent IP address or cookie.

Chapter 3

A Large-Scale Study

3.1 Introduction

The work described in this chapter and Chapter 6, is largely based on [5]. In this chapter we review previous work looking at this issue and we report on a survey we undertook to discover the current situation. In this rapidly evolving area, repeated surveys are vitally important and, as we discuss in this chapter, it appears that fingerprinting has become much more common since previous studies were performed.

A number of authors (e.g. Eckersley [18], Acar et al. [1], Mayer and Mitchell [52] and Nikiforakis et al. [61]) have examined the range of attributes that could be used for browser fingerprinting — a summary of some of these attribute types was given in the previous chapter. Although the range of retrievable attributes, as well as methods for retrieving them, have been widely discussed, relatively little has been published regarding the real-world prevalence of browser fingerprinting, who is deploying it, and the types of attributes collected to achieve it. This issue clearly merits further investigation, and has motivated the work described in the next chapter.

In this chapter we also describe the results of a study of the fingerprinting behaviour of the 10,000 most visited websites. We aimed to discover how many websites deploy active browser fingerprinting, whether directly or through third parties. We also examined and, where possible, identified the attributes that were collected by the sites that were identified as performing browser fingerprinting. One important motive for understanding better the prevalence and nature of browser fingerprinting is to help in developing tools that inform the user about fingerprinting, and also enable users to exert control over the degree to which fingerprinting is possible. To this latter end, in Chapter 6 we describe *FingerprintAlert*, a browser extension developed as part of the study, which makes users aware whenever a website is collecting information usable for

browser fingerprinting. It also allows all detected fingerprinting to be blocked.

The remainder of the chapter is organized as follows. Section 3.2 reviews relevant prior art, and in Section 3.3 we briefly discuss the limitations of this previous work. In Section 3.4 we discuss the main motivations for the study described in this chapter. In Section 3.5 the procedure we used to collect data from 10,000 websites is described; the results obtained are reported in Section 3.7 and are analyzed in Section 3.8. In Section 3.9 we discuss the relationship of our study to the prior art. We summarize the chapter in Section 3.10.

3.2 Previous Work

The following list summarizes (in chronological order) all the previous work which in some way has examined the prevalence of browser fingerprinting. A summary of the findings of these previous studies is given in Table 3.1. As discussed below, many of these papers give only a very limited examination of real-world browser fingerprinting, e.g. by focussing on a single fingerprinting technique.

- In 2013, Nikiforakis et al. [61] attempted to discover the prevalence of browser fingerprinting the top 10,000 websites. This is the first study of this type. They crawled up to 20 pages of each website, searching for specific fingerprinting scripts supplied by three major third-party fingerprinters. The chosen scripts recovered information useful for several browser fingerprinting techniques, including font detection through JavaScript and Flash; and user agent detection through JavaScript and HTTP. The study revealed that fingerprinting is being performed by only 0.4% of the tested websites.

- In 2013, Acar et al. [2] crawled the top one million websites to identify the deployment of a single fingerprinting technique, namely determining the set of installed fonts. They found that 0.04% of tested websites deployed this technique. This was the first study to examine use of the Canvas API for fingerprinting. It was also the first fingerprinting study to examine as many as one million websites and it remains amongst the largest studies of the prevalence of fingerprinting in terms of the number of tested websites.

- In 2014, Acar et al. [1] examined the top 100,000 websites for the presence of three tracking techniques, including a fingerprinting tracking technique using the Canvas API. They found that 5% of the tested websites used this technique. The fingerprinting technique considered in this study differs from the technique

examined in the 2013 Acar et al. study [2] in that it involves canvas-rendered images rather than canvas-rendered fonts.

- In 2015, FaizKhademi et al. [22] described FPGuard, a browser extension they developed that resists fingerprinting. The extension detects various fingerprinting scripts that are indicative of four fingerprinting techniques, namely JavaScript Objects Fingerprinting, JavaScript-based Font Detection, Flash-based Fingerprinting, and Canvas Fingerprinting. The study found that the first three of these techniques are used by the two major fingerprinting providers they examined. The authors used the FPGuard extension on the top 10,000 websites and found that 42.6% of the tested websites appeared to perform fingerprinting.

- In 2015, Libert [47] examined the top one million websites for the presence of third-party JavaScript. He used this as an indication of the *possible* presence of fingerprinting scripts. He found that 83% of the tested websites contained third-party scripts. Libert argued that this might still be a slight underestimate of the real percentage for a number of reasons, including that a small number of websites could be using non-JavaScript scripts/plugins (e.g. Adobe Flash or Microsoft Silverlight).

- In 2016, Engelhardt and Narayanan [20] reported on a study involving crawling the one million most visited websites for the possible deployment of 15 tracking techniques including five widely discussed fingerprinting techniques, namely: use of the Canvas API, detecting the set of installed fonts using the Canvas API, calls to WebRTC, use of AudioContext, and detection of battery charge level using the Battery API. They found a very low rate of use of the last two techniques. On the other hand, they found that Canvas API fingerprinting, font set detection via the Canvas API, and WebRTC-based fingerprinting were being performed by 1.4%, 0.3% and 0.07%, respectively, of the tested websites.

- In 2017, Olejnik et al. [64] crawled the top 50,000 websites to identify the possible use of the battery API for fingerprinting purposes. They found that 1.7% of the tested websites served scripts that accessed this API. This study was the first to examine the use of the battery API for fingerprinting purposes. However, as discussed in 7.4.3, many browsers no longer support this API.

- In 2017, Haga et al. [32] reported on a study which found that 17.3% of the top 100,000 websites were using at least one fingerprinting script. They reached this result by attempting to detect the presence of various known fingerprinting scripts

on the tested websites. They concluded that 50% of the identified suspected fingerprinting is associated with google-analytics.

- In 2018, Das et al. [16] checked whether websites attempted to access motion sensors, which they suggested are mostly used for fingerprinting purposes. They found that 3.7% of the top 100,000 websites attempted to retrieve information through such sensors from visitor devices. It is worth noting that the technique examined in this study can only be used to fingerprint mobile devices. They also found that of the devices which did attempt to retrieve motion sensor data, 63% also gathered other data likely to be useful for fingerprinting purposes.

Table 3.1: Prevalence of fingerprinting according to previous studies

Year	Study	Detection Method	Websites	Fingerprinters%
2013	Nikiforakis et al. [61]	limited scripts	10K	0.4%
2013	Acar et al. [2]	canvas font	1M	0.04%
2014	Acar et al. [1]	canvas	100K	5%
2015	FaizKhademi et al. [22]	various scripts	10K	42.6%
2015	Libert [47]	3rd-party scripts	1M	83%
2016	Englehardt and Narayanan [20]	canvas canvas font WebRTC	1M	1.4% 0.3% 0.07%
2017	Olejnik et al. [64]	battery	50K	1.7%
2017	Haga et al. [32]	various scripts	100K	17.3%
2018	Das et al. [16]	motion sensors	100K	3.7%

3.3 Analysis

As many authors have reported, there are a large number of techniques that could be used for browser fingerprinting. For example, in 2016 Alaca and van Oorschot [6] described 29 different such techniques, and more than 50 methods are demonstrated on the BrowserLeaks website. Moreover, this is a rapidly evolving area, where new techniques are constantly being devised and at the same time other techniques are becoming less effective (e.g. those based on the use of Flash). Thus studies such as those of Acar et al. [2], Acar et al. [1], Englehardt and Narayanan [20] and Olejnik et al. [64], which attempt to detect browser fingerprinting by checking for the use of a small number of fingerprinting techniques, risk seriously underestimating the actual prevalence. It is also interesting to note that all these studies identified possible fingerprinting by examining the downloaded scripts, rather than by looking at the behaviour of these scripts. Interestingly, these studies were the first to identify possible fingerprinting using the Battery and Canvas APIs.

An alternative approach to detecting browser fingerprinting is to detect the presence of certain scripts known to be widely used for the purpose. This is the approach followed by Nikiforakis et al. [61], FaizKhademi et al. [22] and Haga et al. [32]. However, this approach again risks underestimating the actual prevalence, given that the set of scripts is constantly evolving. However, detecting only known fingerprinting scripts has the advantage of not giving any false positives, i.e. providing a lower bound on the level of fingerprinting.

The study of Libert [47] is likely to overestimate the presence of fingerprinting since it is based on the presence of *any* third-party scripts on an examined website. Of course, this approach is still valuable as it provides an approximate upper-bound on the proportion of websites performing active fingerprinting.

Finally, Das et al. [16] report on a study performed after that described in Chapter 4, which looked only at fingerprinting that involved the use of mobile phone sensors. This was the first study to examine the real-world use of mobile phone motion sensors for fingerprinting. Unsurprisingly, the study found that only a relatively small proportion of websites were fingerprinting using this approach.

3.4 Motivation

Despite the fact that browser fingerprinting techniques have been extensively studied, as noted above, relatively limited information is available on its prevalence and the browser attributes that are collected in practice. To the author's knowledge, no study prior to that described in Chapter 4 has attempted to discover all the browser fingerprinting attributes that are collected by a large set of real-world websites. Moreover, all previous studies except Libert [47] identified fingerprinting by the presence of certain scripts. Fingerprinting using any other scripts would therefore have been missed.

These observations motivate the work described in the next chapter, in which we report on a study of the fingerprinting behaviour of the 10,000 most popular websites. Determining which attributes are being used for browser fingerprinting (regardless of scripts deployed) is a key step in trying to understand how best to control it, a key goal of this thesis.

3.5 Data Collection Methodology

3.5.1 Data Gathering

The main objectives of the data collection exercise were to assess the number of websites performing browser fingerprinting, and discover what types of data are being collected

for this purpose. To achieve our objectives, we decided to crawl a large number of well-used websites and to test their data gathering behaviour. We chose 10,000 sites as this seemed both sufficiently many to generate representative results, and also a manageable number so we could analyze the considerable volumes of data generated.

Unlike the work others including Englehardt and Narayanan [20] and Acar et al. [2], we chose not to examine the JavaScript itself, but instead monitor the data that is actually transferred back from the browser. We only looked at the data transmitted, rather than analyzing the downloaded JavaScript, for two main reasons: manual analysis of JavaScript on this scale was infeasible, and detection of certain scripts, as noted in 3.2, has limitations. Moreover, the data that is sent was the key issue of concern for us, not so much how it is gathered. That is, we based our fingerprinting detection on an analysis of the data that is communicated, regardless of the recipient.

We used a simple method to decide whether a web server is performing browser fingerprinting. That is, we looked at the data sent back from the browser to the visited website after the first web page was loaded, and checked whether we could detect any data that is potentially being collected for the purposes of fingerprinting. The precise definition of the data types that triggered our decision is given in Section 3.5.3. To try to "normalize" web server behaviour, we looked only at the interactions that occur when a browser initially visits the homepage of the website, rather than other information gathering exercises that might occur (e.g. when a user tries to log in). We opted only to examine the homepages of tested websites since other studies, such as that of Nikiforakis et al. [61] that involved crawling up to 20 pages of every tested website, failed to find any benefit from visiting pages other than homepages.

3.5.2 Experimental Set Up

In order to select which websites to crawl, we retrieved the top 10,000 websites from the freely available Majestic list of the one million most visited websites[1]. As noted above, we only looked at the interactions that occur when a browser initially visits the homepage of a website, rather than other information gathering exercises that may occur (e.g. when a user tried to log on), in order to try to "normalize" website behaviour. This of course means that we missed websites that employ interaction-triggered fingerprinting.

The crawler was created using Selenium WebDriver[2], a Python script, the *FingerprintAlert* Chrome extension (described in Chapter 6), and the Chrome browser itself. Summaries of the crawler components and the devices used for the study are given

[1] Majestic is a website specializing in web usage statistics, and provides a daily-updated list of the top one million websites, https://majestic.com/reports/majestic-million [accessed on 09/10/2017].

[2] Selenium is open-source software used to automate browsers for testing purposes — see https://www.seleniumhq.org.

in Tables 3.2 and 3.3 respectively, and the structure of the experimental platform is depicted in Figure 3.1. The Python script instructs Selenium to visit the 10,000 websites in the list, wait for each to fully load, and then wait for a further short period before moving to the next website. The code for the Python script is given in Appendix B.

Table 3.2: Crawler software components

Component	**Details**
Browser extension	FingerprintAlert 1.0
Programming language	Python 3.6.3
Automation tool	Selenium 3.8.1

Table 3.3: Computing environment used for crawling

Component	**Details**
Device	MacBook Pro (10.1.1)
OS	MacOS Sierra 12.1
Browser	Chrome 62.0.3202.94

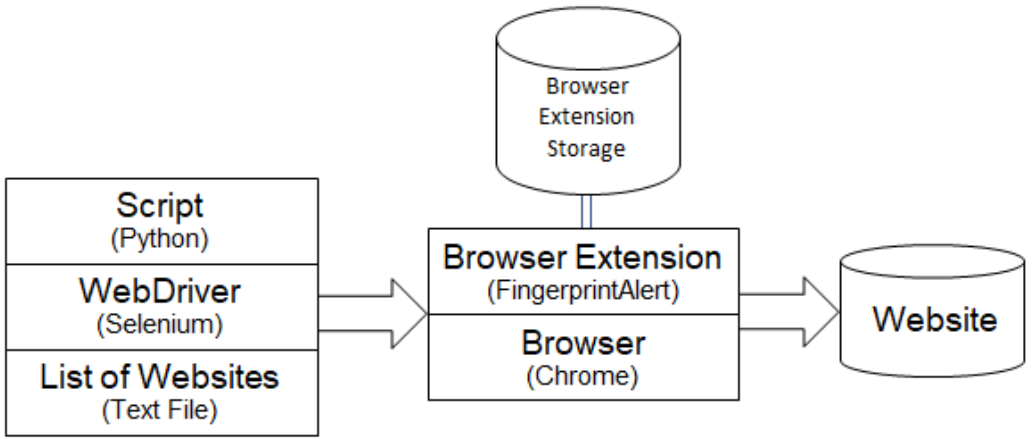

Figure 3.1: Crawler components and their interactions

The delay is included because, in preparatory work, we manually visited 50 websites on the list and found that following the full loading of the page some only relayed information after a delay ranging from one second to several minutes. Such waits seem likely to be both to allow the various elements of the web page to be loaded and executed, and to take account of dynamic content (e.g. advertisements) being continuously loaded.

We set the short delay to three seconds; this was a fairly arbitrary choice, although it was long enough to cause a number of websites to transmit data, although not sufficiently long to make the crawling process significantly more time consuming.

The extension collects and stores all data that is relayed from the browser to one or more web servers using the `GET`, `POST` or `HEAD` HTTP methods[3] [25], i.e. the commonly used means by which information, including attributes used for fingerprinting, is relayed from browser to server. Whether or not the data was sent SSL/TLS-protected, i.e. using HTTPS [69], was also recorded.

The crawling process took approximately 300 hours to complete. It took this long for several reasons, including that some websites took several minutes to fully load, and that Selenium occasionally crashed. In such cases, the crawler was restarted manually, and we re-crawled websites after a crash to ensure we did not miss any data.

3.5.3 Data Processing

Prior to the full crawling process we initially crawled a smaller sample (approximately 1,000 of the websites) to test the crawler. In this process we indiscriminately collected all data sent (if any) from the browser to web servers. Manual examination of the collected data revealed that in many cases it included information unrelated to the visiting device or the browser (e.g. the URLs of displayed advertisements), i.e. of no interest to this study. Most importantly for our purposes, we were also able to identify fingerprinting attributes that had unique formats or values (e.g. screen resolution: `1920x1080`) that made automatic detection possible.

Using these preliminary findings, we programmed our crawler to automatically detect a set of 17 attribute values. To achieve this we first collected the precise values of these 17 attributes for our experimental platform, as summarized in Table 3.4. For example, this included the public IP address in use at the time of the experiments, i.e. `165.120.95.194`. The crawler then used regular expressions to examine all relayed data and match them against the set of 17 attribute values for the experimental platform. If the crawler detected that a website was causing the browser to send any of these 17 attribute values to the same or a different website, then it captured the entire HTTP message and stored it as part of the data collected.

The presence of one or more of the experimental platform attribute values in data returned by the browser was used to determine whether or not a website was engaged in fingerprinting. This set of 17 attribute types includes many of the attributes whose use for fingerprinting is most widely discussed, so we believe that the presence or absence

[3] The quantity of data that can be relayed using `GET` or `HEAD` is very limited, whereas `POST` allows the transmission of very large volumes (megabytes) of data.

of an attribute of one of these types is a reasonable indicator of whether fingerprinting is being performed. However, some other attribute values are much more complex, and hence are difficult to automatically identify. In subsequent manual analysis of the recorded data, we were able to identify many additional attributes because they were labelled by name in the captured data and their values matched those of the browser used in the experiment. To perform this task automatically would have been extremely difficult because some sections of the recorded data were not parsed, and the substrings of the data that were parsed varied in format (unsurprisingly given the absence of any standards for data formats for transferred attribute values).

Table 3.4: The 17 chosen browser attributes and their values for the test platform

Attribute	**Value**
Resolution	`1280x800`
OS	`Mac OS X`
OS Version	`10.12.6`
User Agent	`Mozilla/5.0 (Macintosh; Intel Mac OS X 10_12_6) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/62.0.3202.94 Safari/537.36`
Browser Name	`Chrome`
Browser Version	`C62.0.3202.94`
WEBGL Renderer	`WebKit WebGL`
WEBGL Vendor	`WebKit`
WEBGL Version	`WebGL 1.0 (OpenGL ES 2.0 Chromium)`
GPU	`Intel(R) Iris(TM) Graphics 6100`
GPU Vendor	`Intel Inc.`
Installed Plugins	`Chrome PDF Plugin, Chrome PDF Viewer, Native Client, Widevine Content Decryption Module, Widevine Content Decryption Module`
Language	`en-GB`
Location	`51.4167,-0.5667`
City	`EGHAM`
IP Address	`165.120.98.194`
Character Set	`UTF-8`

In order to manually identify fingerprinting attributes in the collected data we searched for the fingerprinting attributes reported by Alaca and Van Oorschot [6] and

the BrowserLeaks.com website, including attributes not in the list of 17 attribute types detectable by the crawler. We then attempted to match these values with the values in the collected data. Once we completed the matching, we manually inspected the matches found; this was necessary to ensure that the matches found were genuine and not coincidental similarities in strings or numbers. In most cases the match was confirmed by finding labels followed by the expected values in the collected data.

3.6 Challenges and Limitations

We faced a number of challenges in both implementing crawling and processing the collected data. First, websites are unlikely to admit use of browser fingerprinting, and so we can only attempt to judge their behaviour based on the types of information retrieved from the browser, and when it was collected. As discussed in Chapter 2, there is a wide range of attributes that, when put together, can be used to create a unique device fingerprint. Identifying and monitoring all such attributes is very challenging, especially since new attributes seem to arise frequently (given continuously evolving browser functionality). Moreover, many websites cause the browser to send a series of data strings back to the server; automatically, or even manually, identifying what these data represent is highly non-trivial. It was not always possible to parse the data sent since there is no standard for such data transmissions; indeed, some websites may deliberately obfuscate the data they send. It was therefore impossible to fully interpret all the data. Fortunately, there are certain attributes that are easily identifiable because of their special format and range of values, such as screen resolution (e.g. `1920x1080`), fonts (e.g. `Arial`), or geolocation coordinates (e.g. `51.4167, -0.5667`).

It is very difficult to determine the minimum number of attributes needed to produce a unique fingerprint. Fingerprint uniqueness depends on many factors, including the range of values of an attribute, how often it changes, and how much it varies across browsers and host platforms. As a result, we made the simplifying assumption that a website is deemed to be engaging in browser fingerprinting if it causes a browser to send at least one of the 17 attribute values given in Table 3.4.

As noted in Section 3.5.2, the crawler only visited the *homepages* of the chosen 10,000 websites. Websites we reported as not deploying browser fingerprinting might nevertheless still be doing so on other pages. Moreover, the attribute collection reported here was unprompted (i.e. no clicking, cursor movements or typing was involved) except for loading of the web page. Through manual visits to selected websites, we found that some websites only cause the browser to send fingerprinting attribute values when there are further interactions. Moreover, some websites only retrieved attributes when a user

submits a form or logs in, and such cases would be too complex (if not impossible) to capture automatically. The focus of this study is fingerprinting that targets everyone, including those engaged in casual browsing.

As our crawler was Selenium-based, it suffered from the known crashing problem [20] on certain websites, e.g. when it was unable to fully load all the elements of a website. In such cases the crawler had to be manually restarted. On average, Selenium crashed after crawling 155 websites. Moreover, Chrome extensions are limited to 5MB of storage and so, to ensure that the collected data did not reach that limit, we programmed the crawler to stop after every 200 visited websites, yielding an average of 3MB of collected data. However, Selenium usually crashed before reaching the 200-website limit.

The crawler was set to allow a website a maximum of 60 seconds to fully load before it timed out. We found that the tested 10,000 websites took an average of 19 seconds to fully load. Our tests were performed using an Internet connection with a minimum bandwidth of 40 Mbps, and so connection limitations are unlikely to be the reason for the loading delays. The time to load a website noticeably increased as we went through the list of crawled websites, i.e. the less popular websites loaded more slowly. So, in future similar experiments, we would recommend that crawlers should not timeout until at least 20 seconds have elapsed.

3.7 Results

The data collected in this study, as well as the tools we used for data collection and analysis, are available at `https://github.com/fingerprintable`. The dataset includes the contents of all HTTP messages sent by and to the crawled websites that the crawler identified as performing fingerprinting. This includes the data retrieved from the visiting device (i.e. the device used for data gathering), as well as the domain names of the sender and receiver of the data. Figure 3.2 shows a sample of a complete block of data from amongst those collected in our study.

Using a combination of automated parsing and manual inspection, we detected the transmission of 286 different attribute types. We further detected 1,914 distinct websites that collected data for fingerprinting. 70 of the 10,000 websites (i.e. 0.7%) timed out (e.g. because the website did not respond) during the crawling process and thus were fully, or partially, excluded from our findings. Overall, 6,876 (68.8%) of the crawled websites caused the browser on the experimental platform to send data that could be used for browser fingerprinting (either to the same or another website, i.e. first-party or third-party fingerprinting). We refer to such websites as *fingerprinting websites*; of course, despite the name, the fingerprinting websites might not actually be using the

```
{
  "toURL": "<tracker URL is shown here>",
  "referer": "<visited website URL is shown here>",
  "method": "POST",
  "data":
"fp={\"os\":\"2\",\"browser\":\"Chrome61,0,3163,100\",\"fonts\":\"
undefined\",\"screenInfo\":\"1280*800*24\",\"plugins\":\"Portable
Document Format::internal-pdf-viewer::Chrome PDF
Plugin|::mhjfbmdgcfjbbpaeojofohoefgiehjai::Chrome PDF
Viewer|::internal-nacl-plugin::Native Client|Shockwave Flash 24.0
r0::internal-not-yet-present::Shockwave Flash|Enables Widevine
licenses for playback of HTML audio/video content. (version:
     1.4.8.1008)::widevinecdmadapter.plugin::Widevine Content
Decryption Module\"};"
}
```


Figure 3.2: Excerpt of collected data

collected data for fingerprinting.

Fingerprinting is most commonly performed by third-party sites; 84.5% of the 6,876 sites collecting data sent it only to third parties. Of the rest, 2.4% were exclusively first-party fingerprinters, with the other 13.1% using both first- and third-party data collection. Over the 6,876 fingerprinting websites, data was sent to an average of 3.42 domains. The largest number of different data-collecting websites to which data was sent for a single visited website was 42.

Fingerprinting websites collected an average of 1.75KB of data. The third-party websites that collected the most data were yandex[4] (2.9MB), optimizely[5] (2.8MB) and casalemedia[6] (2.1MB). Figure 3.3 shows the top 10 third-party websites in terms of collected data volume for a single visiting browser.

Of the attributes the crawler can automatically detect, the three most frequently collected were: screen/browser resolution, language, and charset (i.e. character encoding). We found that fingerprinters collected, on average, five of the attribute types listed in Table 3.4. Figure 3.4 summarizes the 10 most frequently collected attribute types. The most widely used fingerprinting third-party was google-analytics[7] (see Appendix A.2 for a complete list of fingerprinting third parties); google-analytics provides web analytics as well as other web-based services to websites. This finding is consistent with studies those of non-fingerprinting-based tracking (e.g. of Schelter and Kunegis [73] and Metwalley et al. [54]), which revealed that google-analytics is the most commonly occurring third-party tracker.

[4] https://yandex.ru
[5] https://www.optimizely.com
[6] http://casalemedia.com
[7] https://analytics.google.com

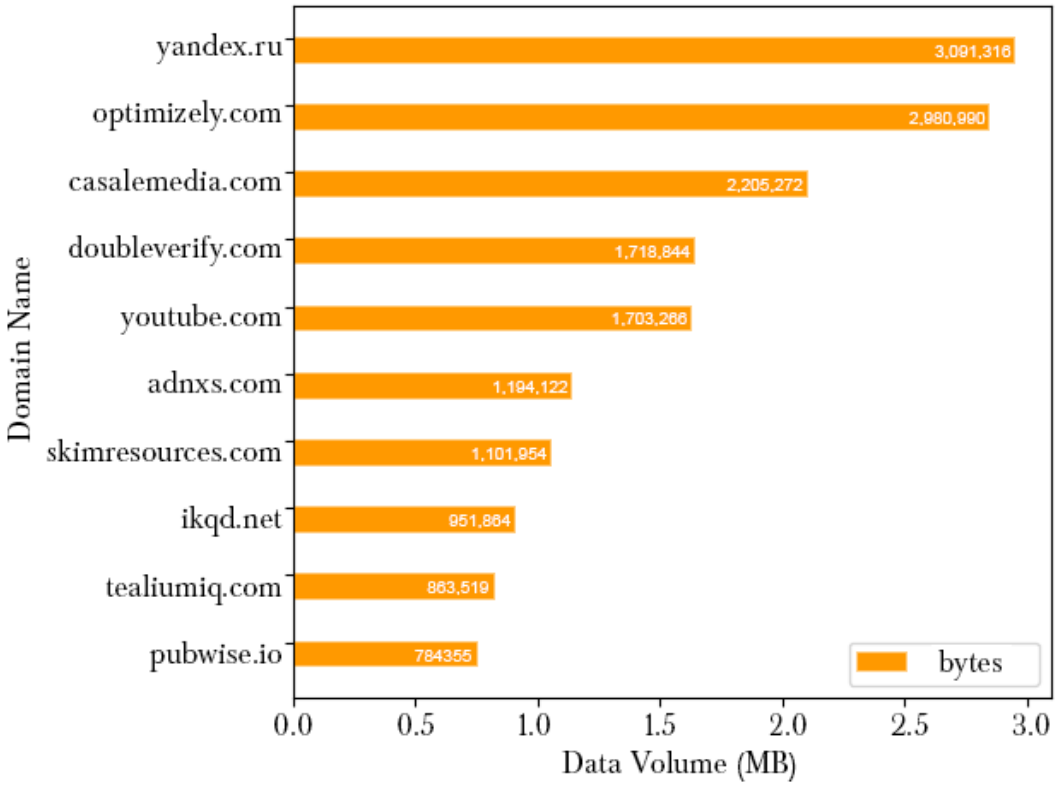

Figure 3.3: Top 10 fingerprinters in terms of collected data volume per browser

As noted above, amongst the collected data we were able to identify 286 fingerprinting attributes, which we divided into six categories (see Table 3.5). The full list of 286 attributes can be found in Appendix C.

Table 3.5: Summary of identified fingerprinting attributes

Attribute Type	**WebGL**	**Features**	**Media**	**IO***	**Network**	**Misc.**	**Total**
Count	114	66	41	20	10	35	**286**

*Input/Output

3.8 Analysis

3.8.1 Processing Collected Data

As described above, the crawler logged every website that relayed data if one, or more, of the 17 pre-programmed attribute values were detected. We examined random samples of the collected data to identify the presence of any false positives. We found some HTTP

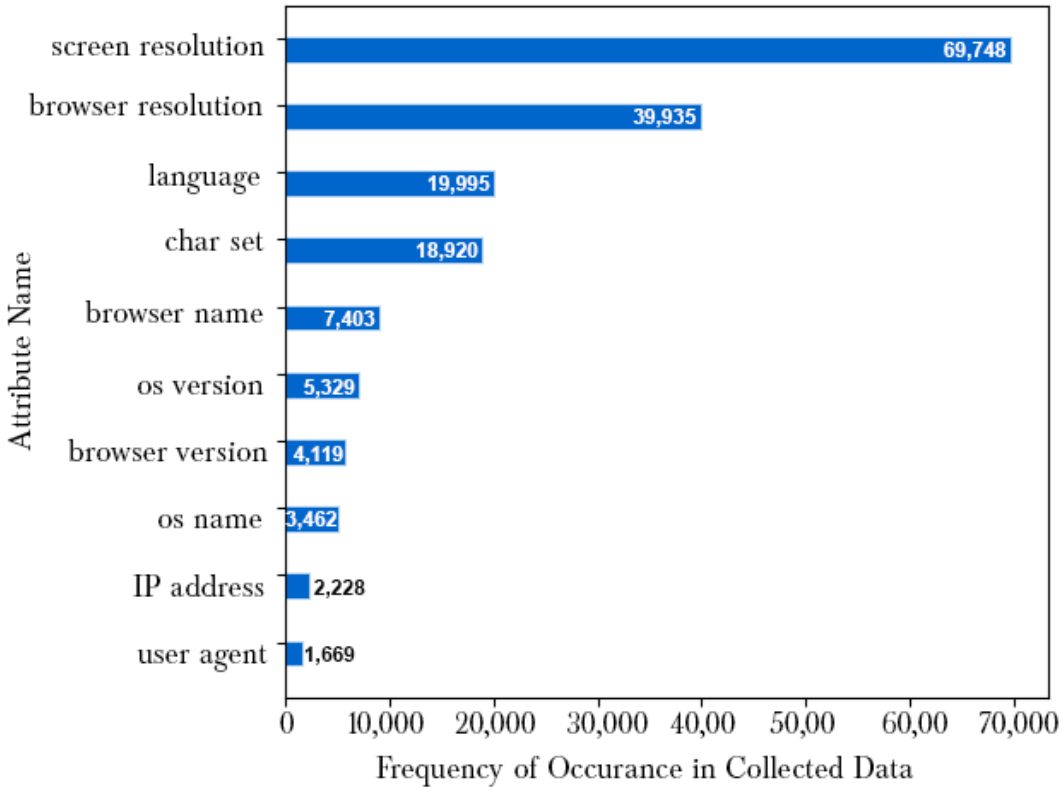

Figure 3.4: Top 10 collected attributes

messages that contained data that were incorrectly matched with one of the 17 attribute values. We wrote a script to remove such records (e.g. if the string `1280088.jpeg` matched with the screen resolution width `1280`). However, in general, identifying false positives (if any) in the filtered data is non-trivial, since the ability to fingerprint browsers typically depends on both the number and type of collected attributes. For example, Mowery and Shacham [56] have demonstrated that the Canvas API alone could be enough to fingerprint a browser, and Laperdrix et al. [44] demonstrated a seemingly successful method of fingerprinting based on a specific set of just 17 attributes.

3.8.2 Prevalence of Fingerprinting

Our study confirms the findings of Englehardt and Narayanan [20] that fingerprinting is commonplace, at least by widely-used websites, and yet there are a relatively small number of entities actually collecting and processing attributes (mainly third-party trackers). Indeed, the top five third-party fingerprinting domains (see Figure 3.5) are all part of a single company, Google Inc. This finding is consistent with Libert [47],

who found that 78.07% of the top one million websites send data to a Google-owned domain. Moreover, it is consistent with Haga et al. [32] who found that google-analytics is, potentially, the most prevalent fingerprinter. An extended list of fingerprinters can be found in Appendix A.

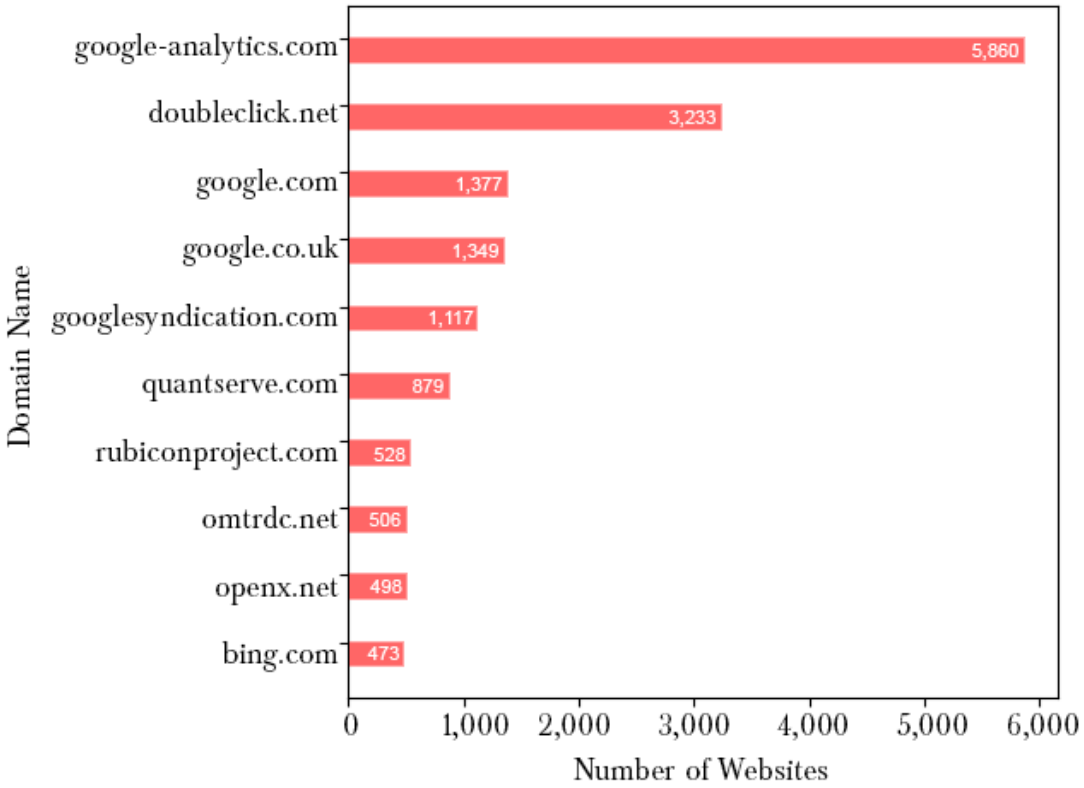

Figure 3.5: Top third-party fingerprinting domains

We found that 68.8% of the top 10,000 websites are potentially engaged in fingerprinting, although, as shown in Table 3.2, previous studies have yielded rather different results. For example, in 2013, Nikiforakis et al. [61] found that only 0.4% of the top 10,000 websites deployed fingerprinting. A year later, Acar et al. [1] reported that 5% of the top 100,000 websites deployed browser fingerprinting using the Canvas API. In 2015, FaizKhademi et al. [22] found that 42.6% of the top 10,000 websites were likely to be engaging in fingerprinting. It thus seems likely that both the prevalence of browser fingerprinting and the number of attributes being collected for this purpose have significantly increased.

3.8.3 Fingerprinting Attributes

As discussed in 3.5.3, we were able to identify 286 fingerprintable attribute values retrievable from the browser used in our experiment. We were able to confirm their presence by comparing attributes in data collected by the crawler with attributes reported by Alaca and Van Oorschot [6] and the BrowserLeaks website. This gave us an indication of the range of attributes that are collected in the real world, as opposed to those discussed in the literature, and also helped us improve the functioning of the extension described in Chapter 6.

It is worth noting that the number of attributes reported in our study is much larger than those reported by previous studies. This is partly explained by the fact that previous studies have searched for a smaller number of attributes; for example Eckersley [18] and Cao et al. [12] looked for just 10 and 53 attributes respectively. The significantly higher number we found also seems likely to be a result of the growing use of browser fingerprinting [2, 61], and the fact that we monitored the HTTP messages transmitted between visited websites and potential trackers as opposed to only detecting the presence of known fingerprinting scripts, as previously widely performed. All of the attributes we were able to identify are collectable by BrowserLeaks.com. However, BrowserLeaks.com can also collect many attributes that we did not find any websites to be collecting, including many of the browser features collectable by *Modernizr*[8].

3.8.4 Deployment of HTTPS

Some fingerprinting websites do not use HTTPS to send the fingerprinting attributes, which are thus transmitted in plaintext; this is a potentially significant user privacy threat. Of the 1,914 distinct fingerprinters we detected, as many as 683 used only HTTP for attribute transmission, 274 mixed use of HTTP and HTTPS, and the remaining 957 used only HTTPS. That is, 50% of the fingerprinting websites used HTTP at least in some cases for transmitting what could be construed as personally identifiable information. Seemingly, the use of HTTP is more common in less popular websites, as Merzdovnik et al. [53] reported that as many as 60% of the top 100,000 websites performing fingerprinting used HTTP. These results apply only to the use of HTTP/HTTPS for transmitting browser attributes, not to whether or not the visited website uses HTTPS. It is interesting to note that we identified a fingerprinting website that used the WebSocket protocol[9] as well as HTTP.

[8] A JavaScript library that help websites detect the availability of CSS and HTML5 features in a visitor's browser `https://modernizr.com`.

[9] WebSocket is a relatively new full-duplex TCP communication protocol [24]. Bashir et al. [8] have described how trackers typically utilize the WebSocket protocol to circumvent ad blockers on browsers.

3.8.5 Fingerprint IDs

Some websites cause a browser to send a value that is explicitly labelled *fingerprint* or *fp*, along with fingerprinting attribute values. These values are typically strings of alphanumerics (e.g. `b5a2ab93-9415-32be-b1ca-c3ddabb110cc`) that appear to function as platform/user identifiers. Evidently, some first- and third-party trackers share such user identifiers [23], allowing them to compile extensive profiles of users. This also means that a website or a tracker could acquire user- or platform-related information without any prior interaction with that user. Such ID-sharing practices clearly make browser fingerprinting-based tracking more privacy-threatening.

3.9 Relationship to the Prior Art

As discussed in 3.2, the study of Libert [47] provides an upper bound on the prevalence of browser fingerprinting at the time it was conducted, because he assumed that *all* third-party scripts could potentiality be used for fingerprinting. Hence, it might be expected that the prevalence of fingerprinting found by this study would be less, given the method used was less indiscriminate, which was indeed the case. A further difference between the work described here and all other previous studies, including that of Englehardt and Narayanan [20], is that they examined the fingerprinting scripts while we examined the data relayed back to server via HTTP. Most significantly, and as discussed in 3.8.2, we not only detected a level of fingerprinting prevalence that was less than that found by Libert [47], but it was much greater than that found by all other studies. Indeed, our results suggest that fingerprinting is becoming ubiquitous. This is consistent with the findings of the large scale study by Lerner et al. [45], that examined the prevalence of online tracking between 1996 and 2016. They have demonstrated an exponential increase in the use of fingerprinting-based tracking in recent years.

Given that this is a rapidly changing and evolving area, it is important to repeat studies frequently, and so one contribution of this study is to reveal the current state of the art. We do not claim that the approach we have adopted is better than other approaches, but it does have the advantage of being based purely on the data itself, and not on the many and various scripts that might be used to fingerprint browsers. Our study has enabled us to give an up to date, fairly comprehensive, and large-scale list of the attributes being used in practice for browser fingerprinting.

Ethical Issues. Clearly any experiment involving real world websites raises potential ethical issues. However, no data relating to individuals were accessed, no vulnerabilities in websites were discovered or exploited, and all websites were accessed as intended by their providers. Websites were crawled only once, except in cases of

a crawler crash where an additional visit was required. All the results are publicly available, as described in Section 3.7.

3.10 Summary

Browser fingerprinting is a relatively new method of uniquely identifying browsers that can be used to track web users. In some ways it is more privacy-threatening than tracking via cookies, as users have no direct control over it. A number of authors have considered the wide variety of techniques that can be used to fingerprint browsers; however, as we have described, relatively limited information is available on how widespread browser fingerprinting is, and what information is collected to create these fingerprints in the real world. There is limited information available in the literature on how widespread browser fingerprinting is at present, and what information is being collected to create these fingerprints. To address this we crawled the 10,000 most visited websites; this gave insights into the number of websites that are using the technique, which websites are collecting fingerprinting information, and exactly what information is being retrieved. We found that approximately 69% of websites are, potentially, involved in first-party or third-party browser fingerprinting. We further found that third-party browser fingerprinting, which is potentially more privacy-damaging, appears to be predominant in practice.

Chapter 4

Comparing Browser Fingerprintability

4.1 Introduction

The work described in this chapter is mostly based on [4]. In this chapter we describe a series of systematic tests performed on currently widely used browsers, which show that some browsers reveal substantially more fingerprinting information than others. Hence users of the least privacy-respecting browsers can more readily be identified and/or tracked. As discussed in Chapter 2, a number of authors have examined the effectiveness of various techniques for browser fingerprinting, but to our knowledge no-one has previously compared the effectiveness of fingerprinting across a range of browsers.

We performed the tests using a specially established website[1]. This website does not retain any data recovered from visiting browsers, but simply displays the information that it is able to collect from the currently employed browser. We hope that this site will be a useful tool in promoting general understanding of the privacy threat arising from browser fingerprinting, and more generally from some of the features provided by today's browsers when executing JavaScript.

As discussed in 2.2, over the last nine years, a number of authors have performed detailed studies of the effectiveness of a range of browser fingerprinting techniques. We used a selection of known fingerprinting approaches to compare the fingerprintability of widely used web browsers on both desktop and mobile platforms. Since desktop browsers differ significantly from their mobile counterparts in their capabilities and

[1] https://fingerprintable.org. All the scripts used in our experiments are publicly available — see Appendix D.

features (e.g. plugins cannot be installed on mobile browsers), we made parallel studies for these two platform types.

The remainder of the chapter is structured as follows. We start in Section 4.2 with the methodology used in our experiments. In Section 4.3 we discuss the experimental results. In 4.4, a summary of the chapter is provided.

4.2 Methodology

We performed our experiments on five of the most widely used platform types. Specifically, we chose to examine browsers running on Windows 10 and Mac OS X 10.12 (Sierra) for desktop platforms, and Android 7.0 (Nugget), iOS 10.2.1 and Windows 10 Mobile for mobile devices (full specifications of the devices used in this experiment are given in Appendices E.2 and E.3). Further details of the methodology we employed to examine browser fingerprintability, including the set of browsers we examined, are given below. Precise details of the versions of operating systems and browsers used are given in Appendix E.1.

4.2.1 Browsers

As noted above, given the major functional differences between desktop and mobile browsers, we made parallel studies of the two classes. For both mobile and desktop platforms, we chose to examine the five most widely used browsers according to netmarketshare.com[2].

- The desktop browsers we examined were **Chrome, Internet Explorer, Firefox, Edge** and **Safari**.
- The mobile browsers used in our tests were **Chrome, Safari, Opera Mini, Firefox** and **Edge**. We excluded Mobile Internet Explorer and Android Browser because they are no longer being developed or included with new devices[3]. Specifically, Google has replaced its native Android Browser with Chrome, and Microsoft has replaced Internet Explorer with Edge in Windows 10 Mobile.

4.2.2 Installation Options

The use of browser extensions and plugins can both increase and decrease the information available for fingerprinting. The presence of extensions inherently increases fingerprinting

[2] https://www.netmarketshare.com/browser-market-share.aspx [accessed on 03/03/2017]

[3] At the time of the study, Internet Explorer and Android Browser were used by approximately 5% of mobile browser users, according to NetMarketShare.

capabilities, since the set of installed extensions (information that is typically available to executing JavaScript) helps individualize a browser; in addition, some extensions reveal information that can identify the user or browser instance (see 2.3.2 and Fiore et al. [27]). On the other hand, as we discuss in Chapter 7, specially designed anonymizing extensions can be used to conceal a browser's fingerprint. To avoid biasing the results, in our tests we used clean installations of browsers so that they did not include any extensions or plugins other than those installed and enabled by default. We could have chosen to disable even those extensions that are present and enabled by default, but we chose to leave them on the basis that many users will not change the browser default settings; hence testing the browser "out of the box" gives the fairest assessment of its privacy properties.

In fact, the browsers we examined come with very few installed and enabled extensions; Edge and Internet Explorer are the only browsers we tested that come with the Flash plugin installed and enabled by default. Although Chrome comes with the Flash plugin installed, it is disabled.

The mobile browsers require various permissions to be set as part of their installation. In addition, browsers may request extra permissions while executing, depending on the features of a visited website (e.g. to request permission to take pictures and record video). For testing purposes, we did not grant any permissions other than those needed for browser installation.

4.2.3 Experimental Scripts

To test the fingerprintability of the selected browsers, a web page containing JavaScript was constructed, intended to be served by our experimental website. Whenever the website is visited by a client browser, e.g. one of those being tested, the scripts in the web page interrogate the browser to learn the values of a set of identifying attributes (as discussed in Section 4.2.4). The scripts used in the experiments were largely based on the BrowserLeaks website as well as those available in the GitHub open repositories.

The web page displayed by the browser contains a summary of the information gathered by the script, and thereby provides an instant summary of the privacy properties of the browser. As mentioned elsewhere, this site is publicly available, and is open for general use. A partial screenshot of a typical displayed page is shown in Figure 4.1.

The total size of the script used is approximately 70KB; in informal tests it loaded and displayed the results without any noticeable delay.

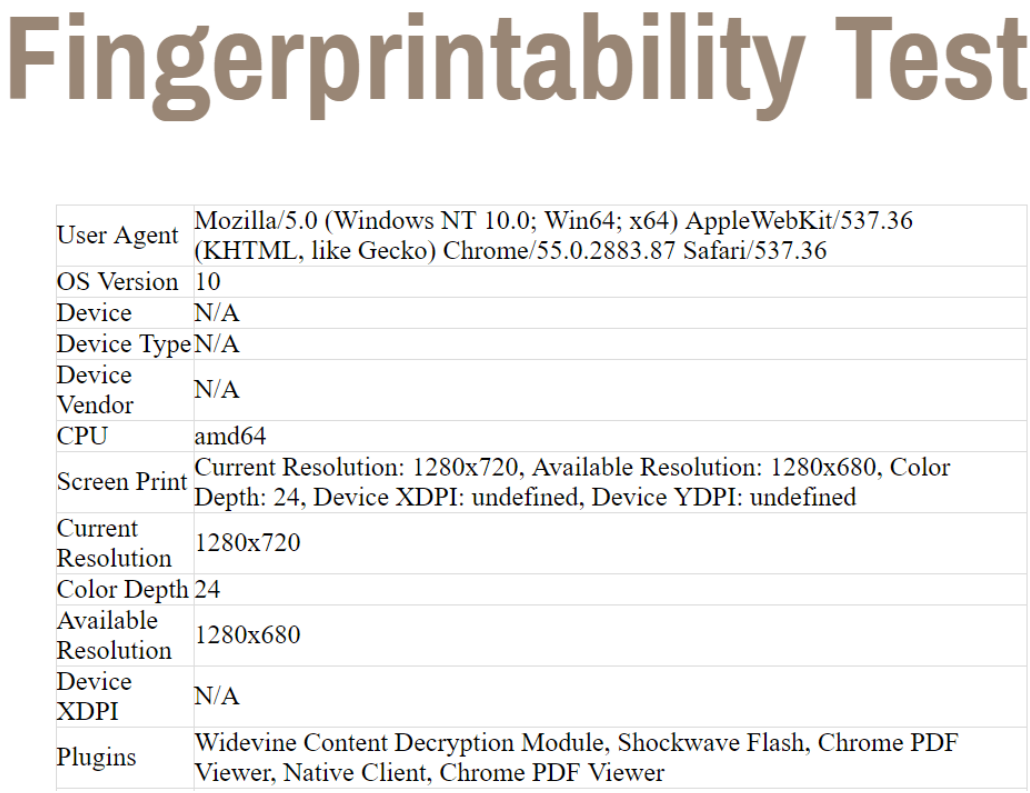

Figure 4.1: Partial screenshot of the Fingerprintability Test page

4.2.4 Attributes

The original goal of our experiments was to sample all the attributes that can be collected from a web browser. Any attribute that is not fixed for all browsers has potential value for fingerprinting. However, a large number of attributes have Boolean values (e.g. Java installed?) or one of a very limited set of values (e.g. Java version) and hence they typically give relatively little identifying information. Given the significant number of such attributes, we therefore omitted all attributes of this type from our tests, and focused on a set of six that have the potential to give significantly more information.

We omitted attributes that, according to Laperdrix et al. [44], take more than a few seconds to collect (e.g. font metrics [26]), or are unreliable for fingerprinting purposes (e.g. battery level [63]). Additionally, we omitted attributes that are made available by all tested browsers as part of their typical functionality (e.g. screen resolution). It is worth noting that some attributes are related to the user's machine and thus can be used to help identify a specific platform even if a user subsequently switches browsers [12]. Others are browser-specific, and hence can only be used for fingerprinting as long

as the same browser is used.

We next discuss in detail the six fingerprinting attributes used in our tests.

Installed Font Set through Flash

As described in 2.3, several methods can be used to identify the set of fonts installed on a device; one involves the use of Flash. If the Adobe Flash plugin is installed and enabled, it can be used to reveal the set, and installation order, of fonts installed on the user platform; this is known to be a highly discriminating attribute (see, for example, Eckersley [18]). Moreover, this attribute can be used to fingerprint a platform even if multiple browsers are used. However, of the desktop browsers we examined, only Edge and Internet Explorer have Flash installed and enabled by default. In this respect, Edge and Internet Explorer are therefore significantly less privacy-protecting than their competitors, since a browser/platform is less identifiable when learning the set of installed fonts is performed without using Flash.

None of the mobile browsers we examined support Flash, so the set of installed fonts was not used when comparing the fingerprintability of this class of browsers. Furthermore, the most widely used mobile operating systems (i.e. Android and iOS) do not give the user the option to install fonts.

Again as described in 2.3, there are other, albeit less accurate, methods of discovering the set of installed fonts using website-supplied JavaScript. However, we do not consider these methods as part of our comparison, since they work in much the same way for all the tested browsers.

Device ID(s)

The use of a device ID as a fingerprintable attribute was proposed by an anonymous developer on BrowserLeaks.com[4]. According to this website, a device ID is a hash value generated by a browser by applying a cryptographic hash function to the unique ID of a hardware component in the user platform (combined with other data values); as described in 2.3 it is retrieved by requesting the WebRTC hardware ID attribute.

The main intended application of such device IDs would appear to relate to managing multimedia content, and the platform components whose identifiers are used are typically the loudspeaker, microphone and/or camera. Since the device ID is computed on other data in addition to a unique hardware identifier, the value computed by a browser will typically change when accessed by different websites. However, for a single website, the device ID appears likely to remain constant (at least for some browsers) across multiple

[4] https://browserleaks.com/webrtc#webrtc-device-id [accessed on 03/03/2017]

visits, giving it high value for fingerprinting purposes. Moreover, a device ID seems to be constant when queried in different ways; for example, we obtained the value via an embedding `iframe` on a different website, and it gave the same value as that for the framed site. In addition to `iframe`, we were able to achieve the same results using other HTML embedding tags, namely `embed` and `object`.

To the author's knowledge, there is no description in the literature of any practical evaluations of this attribute as a technique for fingerprinting, and so its robustness and usefulness for this purpose has yet to be determined. However, experiments conducted as part of this study (see Section 4.3) suggests that it has great promise for use in fingerprinting. This would be an interesting topic for further research. Gaining a better understanding of how exactly the device ID is computed by the various browsers would certainly help in such an investigation, although such information does not appear to be publicly available.

Canvas Image

As described in 2.3, the Canvas API allows a web server to request the return of details of a rendered image, and since different browsers and platforms will render images differently, this can be used for browser fingerprinting. All the tested browsers rendered the sample canvas-rendered image provided by the test script (see Figure 4.2), and as a result give a fingerprint for that browser. We based our tests on the particularly effective canvas fingerprinting approach of Englehardt and Narayanan [20]. Although not part of our comparative experiments, it is interesting to observe that the Tor browser (a modified version of Firefox that uses the Tor network) displays a warning and asks the user for permission before rendering a canvas image. However, none of the tested browsers made such a request.

Figure 4.2: Sample canvas-rendered image used in our tests

Not only does this attribute enable fingerprinting based on the browser in use, but in

some cases it provides the ability to discriminate between two similar platforms running the same browser. That is, in some cases the image rendered by the same browser will differ given a small change in the computing environment.

WebGL Renderer

As described in 2.3, the WebGL Renderer value, as made available via the WebGL API (if supported), indicates either the CPU or GPU of the host platform. This particular attribute contains two sub-attributes: `UNMASKED VENDOR WEBGL` (i.e. CPU/GPU manufacturer) and `UNMASKED RENDERER WEBGL` (i.e. CPU/GPU model).

The experiments revealed that the `UNMASKED VENDOR WEBGL` either states the browser vendor or the CPU/GPU vendor. In both cases it does not provide any useful information that cannot be readily found from the `UNMASKED RENDERER WEBGL` (i.e. identifying a vendor is trivial once the full CPU/GPU model details are known) or the *user agent* HTTP header field readily reveals the browser vendor. We therefore focussed solely on the `UNMASKED RENDERER WEBGL` value.

User Agent

As discussed in 2.3, the *user agent* HTTP header field is a rich source of fingerprinting information. However, we found that all tested desktop browsers provide much the same level of detail, and so this attribute is not useful in comparing their fingerprintability. We therefore do not use this attribute when comparing desktop browsers. However, it remains useful for comparing the fingerprintability of mobile browsers, since in some browsers it includes an indication of the model of the mobile phone.

Private IP Address(s)

Ideally, a website cannot discover the real public IP address of a user platform that is employing a VPN. However, as discussed in greater detail in the next chapter, a website can learn the browser public IP address as well as the IP address assigned by NAT or a VPN by exploiting a feature of its WebRTC implementation. Given its potential for uniquely identifying a platform on its own, we included this attribute in the tests.

4.2.5 Performing the Experiments

Platforms of the specified types, running the chosen operating systems, were equipped with the relevant browsers (clean installs, as discussed above). The browser was then made to visit the test website (`https://fingerprintable.org`) and the data generated by the script was collected and recorded. The 10 datasets (five for the desktop platform

and five for the mobile platform) generated were then processed and used to derive the information given in Section 4.3 below.

Attribute Processing

Each browser was tested for the retrievability of discriminating information for each of the six fingerprinting attributes described in Section 4.2.4 (except for the user agent on desktop platforms and the set of installed fonts on mobile devices). For most attributes, it was straightforward to determine whether or not the browser returned any fingerprintable values. However, some attributes required some processing to be useful. For example, an attribute such as user agent always returns a string of information. The key difference between one browser and another was whether it included information specific to the system hosting it. These differences were observed and noted.

The device ID was tested for both its existence as well as its persistence. We observed that the browsers that calculate such a value, at some point calculate a new value. The main difference between browsers in this respect is in the nature of the trigger that causes recalculation. This means that some browsers have a more persistent device ID, i.e. one that is more valuable for fingerprinting, than others.

In most cases the returned canvas-rendered image was the same for a single platform regardless of the browser. Some browsers also render the same image when running on two devices that have relatively similar specifications. To find these cases we tested and compared canvas-rendered images of each browser on two devices with similar hardware.

Fingerprintability Index

For comparison purposes, we ranked each attribute as having a *high* (3), *medium* (2) or *low* (1) *attribute fingerprintability*, where high indicates an attribute giving more information useful for fingerprinting. These assignments are based on previous work as well as our own qualitative estimations. We have refrained from using the term *entropy* or precise entropy values taken from the prior art, as such values are not available for all the attributes we considered in our study. It is important to note that, regardless of the ranking of attributes, all attributes in our study provide relatively high entropies, as explained earlier in the chapter. Based on the study results, we assigned each tested browser a *Fingerprintability Index (FI)*, which is defined to be the sum of the attribute fingerprintability values of the six attributes we tested; this gives as a simple rough measure of the fingerprintability of the browser.

The fonts attribute is ranked as *high* as it is a highly discriminating piece of information [18]. Device IDs also have the potential of being highly discriminating;

however, as discussed earlier, browsers that provide device IDs differ in terms of the persistence of the values. This attribute is therefore assigned *high* if the browser shows no signs of changing this value under typical browser usage, and is assigned *medium* if a browser provides a new value with every browsing session. It is assigned *low* if a browser provides a new value with every visit or page refresh.

We rank the Canvas API attribute as *medium*, based on the analysis of Laperdrix et al. [44]. However, we rank it as *low* for any browser that returns the same image on two devices with similar specifications. The WebGL information is ranked as *low*, as Alaca and Van Oorschot [6] argue that it provides relatively little information useful for fingerprinting. This is to be expected since many devices could be using an identical CPU and/or GPU.

The user agent reveals a lot of information valuable for fingerprinting [6, 12, 44]. However, we rank it only as *medium* since we focus here purely on whether or not it includes information on the mobile phone model. We assign a rank of *low* to the leaking of private IP addresses. This is because most clients are assigned local IPv4 addresses in the 192.168.0.x range [6] and such IP addresses tend to be dynamically assigned and so can change regularly. However, we assign a *medium* ranking to any browser that reveals one, or more, of the client's ULAs in addition to the aforementioned IPv4 address.

4.3 Results

We next summarize the results of our experiments. We divide the discussion into two parts, first addressing the tests on desktop platforms and second the experiments using mobile devices.

4.3.1 Desktop Browsers

Overview

We summarize below the key observations arising from our examination of desktop browsers.

- **Chrome** did not reveal the set of installed fonts through Flash probing despite the presence of the Flash plugin; this was because the plugin is disabled by default. Chrome is unique in generating a very discriminating device ID. The value remained the same for at least a month, and seems unlikely to change until the browser cache is cleared.

  Canvas image rendering in Chrome resulted in the same hash value on both test machines. We made the same observation on the mobile version of Chrome.

When probed for the WebGL attribute values, Chrome gave the *full* details of the GPU, including the name and version of the installed graphics API. The local IPv4 and temporary IPv6 addresses were also revealed.

- Because **Internet Explorer** comes with the Flash plugin installed and enabled by default, it can be used to determine the set of installed fonts. Internet Explorer does not disclose any device IDs (due to its lack of WebRTC support). The hash value of the image produced using the Canvas API was the same as that generated by Edge on both test machines. Internet Explorer revealed the specific model of the CPU. However, Internet Explorer did not reveal the local IP address.

- **Firefox** does not include the Flash plugin, and hence it does not reveal the list of installed fonts through Flash probing. It generated device IDs, although this attribute is less discriminating than in Chrome as the device IDs change with every browser session.

  Firefox produced two different hash values for the canvas-rendered images on the two test machines. The WebGL probing yielded *Mozilla* as both the vendor and renderer (i.e. neither CPU or GPU were revealed). However, Firefox revealed the client's local IPv4 address.

- **Safari** revealed no information for most of the attributes we tested. This is mainly because of its lack of full WebRTC support and the absence of the Flash plugin. However, it does support the Canvas API and produced the same image hashes on the two test devices. Safari also revealed the WebGL renderer details.

- Just like Internet Explorer, **Edge** comes with the Flash plugin installed and enabled by default. This reveals the set of fonts installed on the computer, which is highly valuable for fingerprinting. Edge generates device IDs but they change every time a website is revisited or even refreshed. It gave the same canvas hash value as Internet Explorer on the test machines. It also revealed the CPU model. Moreover, amongst tested desktop browsers, it was unique in exposing three IP addresses, namely the client's local IPv4, IPv6 and ULA.

Discussion

The results of our tests are summarized in Table 4.1. Only Chrome, Firefox, and Edge provided device IDs. The fingerprintability of this attribute varies significantly between tested browsers. Chrome device IDs are consistent and do not change unless the user

Table 4.1: Desktop browser fingerprintability

Attribute / Browser	Chrome	Internet Explorer	Firefox	Safari	Edge
Fonts	-	●●●	-	-	●●●
Device ID	●●●	-	●●	-	●
Canvas	●	●	●●	●	●
WebGL Renderer	●●	●	-	●	●
Local IP Address	●●	-	●	-	●●●
Fingerprintability Index	**8**	**5**	**5**	**2**	**9**

●= low; ●●= medium; ●●●= high

selects the *private browsing mode*[5] feature or clears the browser cache. The Firefox device ID remained the same during multiple visits in a single browsing session, but changed once the browser was reopened. Of the browsers generating device IDs, Edge gave the value that changed most readily; merely refreshing a web page caused Edge to generate a new value. This makes this attribute in Edge of very limited use for fingerprinting.

All the tested browsers support the Canvas API and rendered the scripted image in our test, i.e. they all reveal this fingerprinting attribute. However, in the case of Firefox, the image resulted in a different hash when rendered on the two test machines. As a result, the canvas-rendered images attribute is more fingerprintable in Firefox than the other tested browsers.

With the exception of Safari and Internet Explorer, all the tested browsers exposed the client's local IPv4 address. Both Edge and Chrome also revealed the IPv6 address. However, Edge was the only tested browser to reveal the client's ULAs. Overall, Edge was the most fingerprintable (FI: 9) and Safari the least (FI: 2).

4.3.2 Mobile Browsers

Overview

Summarized below are the main observations arising from our examination of mobile browsers.

- The **Chrome** user agent revealed the specific phone model. Just like its desktop counterpart, Chrome provided persistent device IDs. Chrome's rendering of the canvas image resulted in the same hash on both testing devices. It also revealed the vendor and model of the GPU, as well as the local IPv4 and ULA addresses.
- **Safari** mobile did not reveal much fingerprinting information except the informa-

[5]Chrome did not assign a device ID when private mode was enabled.

tion derivable from rendering the canvas image and the CPU model through the WebGL API. It rendered the same canvas image on both test devices.

- The **Opera Mini** user agent revealed the phone model. It provided device IDs that were similar to Firefox in terms of calculating a new value with every new browsing session. It also rendered unique canvas images on tested devices. Moreover, it revealed the GPU model, as well as the local IPv4 and ULA addresses.
- **Firefox** did not reveal the phone model in the user agent field, which makes this attribute significantly less revealing. However, Firefox did provide device IDs in the same way as its desktop counterpart. It also rendered unique canvas images on tested devices and allowed the retrieval of the client's local IPv4 and ULA addresses. The WebGL did not reveal the vendor nor renderer.
- The **Edge** user agent included the model of the phone. Edge provided device IDs but, like its desktop counterpart, the IDs change with every page refresh or revisit. It also revealed the model of the GPU. The canvas-rendered image was the same on both test devices. Unlike its desktop version, Edge did not expose any private IP addresses.

Table 4.2: Mobile browser fingerprintability

Attribute / Browser	Chrome	Safari	Opera Mini	Firefox	Edge
User Agent	●●	-	●●	-	●●
Device ID	●●●	-	●●	●●	●
Canvas	●	●	●●	●●	●
WebGL Renderer	●	●	●	-	●
Local IP Address	●●	-	●●	●●	-
Fingerprintability Index	**9**	**2**	**9**	**6**	**5**

●= low; ●●= medium; ●●●= high

Discussion

The results of our tests are summarized in Table 4.2. Chrome, Opera Mini and Edge included the phone model as part of the user agent field. With the exception of Safari, all tested browsers calculated device IDs.

Although all tested browsers rendered the canvas image, Chrome, Safari, and Edge (both desktop and mobile) rendered exactly the same image on the test devices with similar specifications. This makes Chrome, Safari and Edge canvas-rendered images less fingerprintable than the other tested browsers.

Chrome, Opera Mini, and Firefox exposed the local IPv4 addresses. However, unlike their desktop counterparts, they also exposed the client's ULA(s). Overall, Chrome and Opera Mini were the most fingerprintable browsers (FI: 9). Just like its desktop counterpart, Safari was the least fingerprintable (FI: 2).

4.3.3 Other Remarks

It seems reasonable to expect that browser fingerprinting based on the Flash plugin will soon become irrelevant given the imminent disappearance of Flash [44]. In regards to the Canvas API, it is important to note that it is not the rendering aspect of the Canvas API that endangers user privacy but the ability to retrieve details of the rendered image by visited websites. Thus, if this feature was removed from the Canvas API, it would eliminate any possible fingerprinting based on it (at least using current methods). This is an issue that we revisit in a more general context in Chapter 7.

Device IDs have the potential to seriously endanger user privacy, especially given their persistence in Chrome. Moreover, Chrome's persistent device IDs seem unnecessary, given that Edge constantly provides new values.

4.4 Summary

Collectively unique and hence identifying pieces of information, making up what is known as a fingerprint, can be collected from browsers by a visited website, e.g. using JavaScript. However, browsers vary in precisely what information they make available, and hence their fingerprintability may also vary. In this chapter, we reported on the results of experiments examining the fingerprintable attributes made available by a range of modern browsers. We tested the most widely used browsers for both desktop and mobile platforms. The results revealed that Safari was the least fingerprintable browser in both mobile and desktop devices. On the other hand, Chrome was the most fingerprintable mobile browser, while Edge was the most fingerprintable desktop browser. The results revealed significant differences between browsers in terms of their fingerprinting potential, meaning that the choice of browser has significant privacy implications.

Chapter 5

IP Address Compromise through Browser Fingerprinting

5.1 Introduction

The work described in this chapter is largely based on [3]. This chapter focuses on a potentially rich source of browser fingerprinting information arising from a feature of the WebRTC API. We evaluated the availability of this source in a number of browsers. Since the information leaked consists of one or more IP addresses, this leak is especially significant when a VPN is in use, and hence we also tested a number of different VPNs.

Ideally, when a user connects to the Internet via a Virtual Private Network (VPN), the IP addresses (e.g. the public IP address) of the client device are hidden from visited websites. However, as described in 2.3, the client's public IP address can be retrieved by successfully pinging a STUN/TURN server through a visiting WebRTC-supporting browser. If a user is using a VPN for anonymity reasons, then revealing one, or more, of the client's IP addresses to a visited website (or any browser extension that can execute JavaScript on the client's browser) is likely to negate one major purpose of VPN use. Revealing client IP address(es) could also help enable tracking and/or identification of the client as part of browser fingerprinting. Moreover, by using geolocation lookup, a client's public IP address could disclose its country and city [39].

In this chapter, we describe experiments performed to examine five types of client IP address that could be revealed via WebRTC functionality. We also examined to what degree the choice of browser, VPN service and VPN client-side configuration affects the number and type of leaked addresses. A related investigation has been described by Perta et al. [67], who observed the role of the VPN service in IP address leaks. However, they focused only on IPv6 address leaks without looking at other types of IP address or

the role of the browser in the leaks. Moreover, the address leaks they considered are apparently not WebRTC-related. This is the first study to examine all the types of IP address that could leak, as well the first to consider the role of the browser in these leaks.

It is important to note that a WebRTC leak could damage client privacy even if a VPN is not in use. This is because the client private IP address could be leaked, a piece of information which would not otherwise be available to a visited website even in the absence of a VPN. However, these addresses are not necessarily very privacy-sensitive, since clients are typically assigned private IPv4 addresses in the 192.168.0.x range [6].

The remainder of the chapter is structured as follows. In Section 5.2 we discuss the types of IP address that could potentially be leaked via WebRTC. We review prior work related to WebRTC leaks in Section 5.3. The research methodology employed as well as details of the experiments performed are discussed in Sections 5.4 and 5.5. In Section 5.6, we report on and analyze the results of these experiments. Before summarizing the chapter in Section 5.9, we discuss WebRTC leak countermeasures in Section 5.7 and briefly mention disclosure issues in Section 5.8.

5.2 IP Addresses At Risk

In the experiments (see Section 5.5) we found that WebRTC functionality can be exploited to reveal one or more of five types of client IP address, as listed below. Note that the public IPv4 address of a client is not in the list, as in the experiments we performed we were never able to learn such an address using WebRTC.

- *Public IPv6 address*: this is the IPv6 address of the platform and is typically assigned by the ISP of the client.
- *Public Temporary IPv6 address*: this address is assigned by the network to which the client platform is attached.
- *Unique local address (ULA) assigned by LAN*: this IPv6 address is assigned by the network to which the client platform is attached, and is the approximate IPv6 counterpart of the *Private IPv4 address assigned by LAN* [34].
- *Private IP address assigned by the VPN server*: this private (IPv4 or IPv6, depending on the VPN configuration) address is assigned by the VPN server.
- *Private IPv4 address assigned by LAN*: this address is assigned by the network to which the client platform is attached.

It is important to note that due to differences in length and uniqueness, the disclosure of an IPv6 address is more privacy-damaging than that of the private IPv4 address. Moreover, the Public IPv6 address on the experimental platforms remained the same throughout more than two months of testing, while the temporary IPv6 address changed with every connection instance. While the persistence of an IP address depends on the client and network configuration, it seems clear that the public IPv6 address is more persistent than a temporary IPv6 address (hence the name temporary).

More generally, the degree to which the disclosure of a particular type of IP address degrades user privacy depends on its uniqueness and persistence. For example, a private IPv4 address (4 bytes) is typically in the 192.168.0.x range, and is thus far less privacy-sensitive than a public IPv6 address (16 bytes). Moreover, a leak of the IP addresses of clients that are assigned static (i.e. fixed) IP addresses will be more privacy-compromising than if these addresses are dynamically assigned (i.e. they change regularly).

5.3 Previous Work

WebRTC leaks (see also 2.3.2) have been discussed by a number of authors — see, for example, [6, 20, 35, 37, 48, 67]. Of this prior art, Jakobsson [37] explores WebRTC leaks in the greatest depth, but focuses only on public IP address leaks.

Alaca and Van Oorschot [6] observed that WebRTC features could enable a visited website to learn the IP addresses assigned to all the network interfaces of a client platform, including the private IP addresses assigned by a VPN. They deemed this possibility to be a medium-level threat, which seems a reasonable evaluation given they only observed the possibility of private IP address leaks. However, they state in their evaluation that the WebRTC leak issue requires further study.

Englehardt and Narayanan [20] consider the WebRTC threat in some depth, but like many other authors they also only examine private IP address leaks. Liu et al. [48] also only examine leaking of private IP addresses. They claim that the WebRTC issue is only applicable to Chrome and Firefox, but it is not clear what browsers and which versions they tested (the results we obtained, described in Section 5.5, contradict this claim).

Hosoi et al. [35] point out that public IP addresses could be amongst those leaked. As previously mentioned, Perta et al. [67] explore IPv6 address leaks in detail. They report on the results of IP address leak tests of 14 VPN services; however, they do not describe the role of WebRTC in these leaks. A recent IETF Internet Draft [81], details browser mechanisms that can potentially prevent WebRTC-related IP address leaks.

In summary, a number of authors have examined the WebRTC issue, but none have made a comprehensive survey of the issue; typically they have either only examined some of the possible IP addresses that can be leaked, or they have not considered the roles of both the browser and the VPN service in affecting the magnitude of the leaks. In the remainder of this chapter we describe the results of the first comprehensive study of the WebRTC leak issue, including examining the roles of the browser, VPN service and VPN configuration in affecting the nature and volume of IP addresses leaked. This enables us to make recommendations to end users on how they might optimize their behaviour to minimize their loss of privacy. We have also provided a website which enables users to test the privacy properties of their own current browser and VPN configurations.

5.4 Experimental Methodology

We used a modified version of Roesler's publicly available JavaScript[1] to perform the experiments (see Appendix F for the code). The modification incorporates some of the features provided by BrowserLeaks.com that enable the script to work with Edge, which Roesler's original script does not support. Preliminary tests revealed that the number and type of leaked addresses are affected by the choices for both the web browser and the VPN service. We therefore tested five different widely used VPN services running on eight different browser-OS combinations, namely four browsers each running on Windows[2] and macOS[3]. Since we had no information regarding the VPN services that are most widely used, we informally selected five of the top search results in Google. The VPN services we chose to examine are: Hide My A**! (HMA!), ZenMate, ExpressVPN, VyprVPN and TorGuard. Full details of the experimental platforms and VPNs used in the experiments can be found in Table 5.1. Details of the precise versions of the browsers we tested are given in Table 5.2.

We chose to examine the five most widely used desktop browsers according to netmarketshare.com[4], namely **Chrome, Firefox, Edge, Safari** and **Opera**. Although Internet Explorer is the second most widely used browser, we excluded it from the study because it does not support WebRTC and so is not affected by the leaks discussed in this chapter. Moreover, it has been replaced by Edge as the default browser in Windows. Since Chrome, Firefox and Opera are available on Windows and macOS, we tested these three browsers on both operating systems.

[1] https://github.com/diafygi/webrtc-ips [accessed on 17/04/2019]
[2] Windows 10.0.14393 (Build 14393)
[3] macOS 10.12.4 (16E195)
[4] https://www.netmarketshare.com/browser-market-share.aspx [accessed on 14/05/2017]

Table 5.1: VPN program versions

VPN / Specs	Windows	MacOS	URL
HMA!	3.4.6.1	2.2.7.0	https://hidemyass.com
ZenMate	3.4.7.17	1.5.4	https://zenmate.com
ExpressVPN	6.0.9	6.3.3	https://expressvpn.com
VyprVPN	2.9.6.7227	2.14.0.5485	https://goldenfrog.com/vyprvpn
TorGuard	0.3.69	0.3.69	https://torguard.net

Table 5.2: Browser versions

Browser	Version
Chrome	58.0.3029.110
Firefox	53.0.2
Edge	38.14393.1066.0
Opera	45.0.2552.635
Safari	10.1 (12603.1.30.0.34)

Most of the tested VPN programs provide means for the users to modify certain VPN configurations, for example, to switch from one VPN protocol (e.g. L2TP) to another (e.g. PPTP). We found that in some cases this also affects which IP addresses are leaked. This seems likely to be because of the VPN server configuration rather than the protocol itself. Nevertheless, this fact is important to recognize and so we indicate in our results summary below the VPN programs that exhibited such differences (see Table 5.3 for details of the tested VPN configurations).

5.5 Details of Experiments

To perform the experiments, a website (`https://fingerprintable.org/webrtcleaks`) was specially established. The web page contains JavaScript that, when executed in a client browser, fetches all the IP addresses it can retrieve using WebRTC; the retrieved IP addresses (if any) are then displayed on the page (see Figure 5.1). When using it for the tests, the visiting device used either the Windows `ipconfig` command in the command prompt, or `ifconfig` in macOS terminal to identify the types of address displayed on the page.

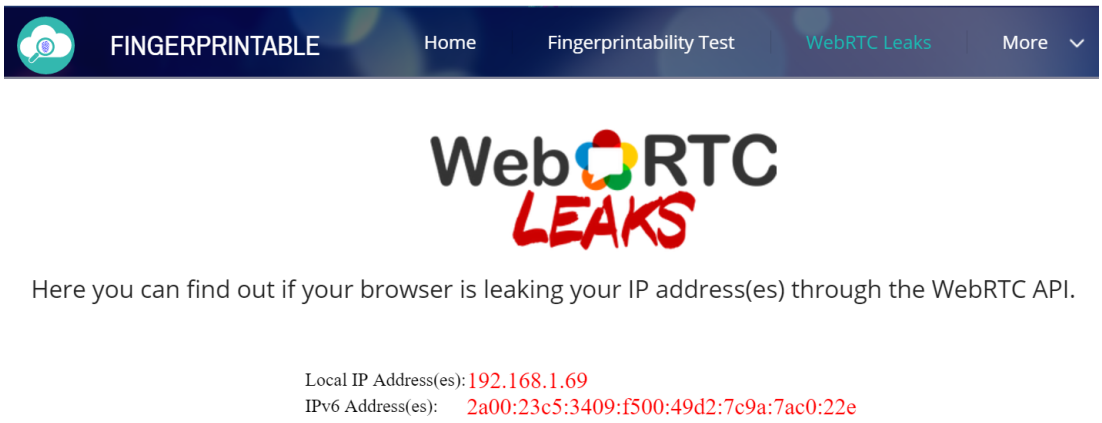

Figure 5.1: WebRTC leak detector

We deployed all five of the chosen VPN programs with each of the eight selected browser-OS combinations, giving a total of 40 test cases. In each case, we caused the client to visit the test page on the specially established website, and documented the IP address(es) displayed. For each of the 40 (VPN, OS, browser) combinations, we visited the test page using all the protocols/configurations supported by the VPN to detect any differences in leaked IP addresses, i.e. for each of the 40 test cases we made between one and five tests (for full details see Table 5.3), giving a total of 116 tests. For example, for each browser-OS combination we visited the test page using VyprVPN a number of times, once using L2TP/IPsec once using PPTP, and so on.

Table 5.3: Tested VPN program configurations

OS / VPN	HMA!	ZenMate	ExpressVPN	VyprVPN	TorGuard
Windows	OpenVPN UDP OpenVPN TCP	N/A	OpenVPN UDP OpenVPN TCP L2TP/IPsec PPTP SSTP	Chameleon* OpenVPN L2TP/IPsec PPTP	OpenVPN UDP OpenVPN TCP OpenConnect UDP OpenConnect TCP
macOS	OpenVPN PPTP	N/A	OpenVPN UDP OpenVPN TCP L2TP/IPsec	Chameleon* OpenVPN L2TP/IPsec	OpenVPN UDP OpenVPN TCP OpenConnect UDP OpenConnect TCP

*VyprVPN proprietary protocol

The tested VPN services provide access to VPN servers in a range of countries.

However, in a series of informal tests we found no difference in the set of leaked IP addresses when connecting to VPN servers for the same service in different countries.

5.6 Results and Analysis

Tables 5.4 and 5.5 summarize the experimental results for Windows and macOS, respectively. Listed in the tables are the types of IP addresses leaked in each test environment. The VPN protocols are also given in the cases where the choice of protocol made a difference to the set of leaked IP addresses. It is worth noting that tests on macOS while deploying a VPN did not reveal a client private IPv4 address, public IPv6 address or ULA.

Table 5.4: Results of experiments on Windows

VPN / Browser	Chrome	Firefox	Edge	Opera
Without VPN	pvt. IPv4 IPv6	pvt. IPv4	pvt. IPv4 IPv6 ULA	pvt. IPv4 IPv6
HMA! (all protocols)	VPN IPv4	VPN IPv4	VPN IPv4 pvt. IPv4 IPv6 ULA	VPN IPv4
ZenMate	VPN IPv4 temp. IPv6	VPN IPv4	VPN IPv4 pvt. IPv4 IPv6 ULA	VPN IPv4 temp. IPv6
ExpressVPN (all protocols)	VPN IPv4 temp. IPv6	VPN IPv4	VPN IPv4 pvt. IPv4 IPv6 ULA	VPN IPv4; temp. IPv6
VyprVPN (Chameleon & OpenVPN)	VPN IPv4 temp. IPv6	VPN IPv4	pvt. IPv4 IPv6 ULA	VPN IPv4 temp. IPv6
VyprVPN (L2TP/IPsec & PPTP)	temp. IPv6	no leak	pvt. IPv4 IPv6 ULA	temp. IPv6
TorGuard (OpenVPN)	VPN IPv4	VPN IPv4	pvt. IPv4 VPN IPv4	VPN IPv4
TorGuard (OpenConnect)	VPN IPv4	VPN IPv4	no leak	VPN IPv4

IPv6 = public IPv6 address; **temp. IPv6** = public temporary IPv6 address; **ULA** = unique local address; **VPN IPv4** = private IP address assigned by VPN server; **pvt. IPv4** = private IPv4 address assigned by LAN.

Table 5.5: Results of experiments on macOS

VPN / Browser	Chrome	Firefox	Safari	Opera
Without VPN	pvt. IPv4 IPv6	pvt. IPv4	no leak	pvt. IPv4 IPv6
HMA! (PPTP)	VPN IPv4	VPN IPv4	no leak	VPN IPv4
HMA! (OpenVPN)	VPN IPv4 temp. IPv6	VPN IPv4	no leak	VPN IPv4 temp. IPv6
ZenMate	VPN IPv4 temp. IPv6	VPN IPv4	no leak	VPN IPv4 temp. IPv6
ExpressVPN (OpenVPN)	VPN IPv4 temp. IPv6	VPN IPv4	no leak	VPN IPv4 temp. IPv6
ExpressVPN (L2TP/IPsec)	VPN IPv4	VPN IPv4	no leak	VPN IPv4
VyprVPN (Chameleon & OpenVPN)	VPN IPv4	VPN IPv4	no leak	VPN IPv4
VyprVPN (L2TP/IPsec)	no leak	no leak	no leak	no leak
TorGuard (all protocols)	VPN IPv4	VPN IPv4	no leak	VPN IPv4

temp. IPv6 = public temporary IPv6 address; **VPN IPv4** = private IP address assigned by VPN server; **pvt. IPv4** = private IPv4 address assigned by LAN.

5.6.1 VPNs

The choice of VPN service had a significant effect on the number and type of IP addresses leaked. In some cases, using one VPN protocol (e.g. L2TP/IPsec) in a VPN program leaked a different number of addresses than another protocol in the same application. We observed no differences in address leakage when switching between the TCP and UDP network protocols, where such options were available. However, when testing different VPN programs, variations in the sets of leaked IP addresses were observed even when the same protocol was in use. We therefore concluded that these differences can be attributed to how the VPN service is configured to handle the connection when using particular protocols.

As can be seen from the tables, TorGuard proved to be the least privacy-compromising VPN service. In all test cases, it revealed none of the client's public IP addresses. At the other extreme, VyprVPN and ExpressVPN did not prevent any of the WebRTC leaks.

5.6.2 Browsers

Safari revealed no IP addresses regardless of which VPN was in use. It is important to note that at the time of the study Safari did not *fully* support WebRTC, as revealed by the (WebKit) specifications status page[5]. However, with the release of Safari 12 in 2018, WebRTC is now fully supported. We re-tested the new version without a VPN service and it still did not reveal any IP addresses; this finding has been confirmed by

[5] https://webkit.org/status/ [accessed on 14/05/2017]

Hazhirpasand and Ghafari [33]. By contrast, Edge revealed four of the five IP addresses discussed in this chapter (only the temporary IPv6 address was not leaked). Edge was the only browser to reveal the public IPv6 address(es) and ULA(s). This makes it the most privacy-damaging browser. This might be because at the time of the study (April 2017)[6] Edge was the only browser that supported ORTC (Object Real-Time Communications)[7], the next generation WebRTC API.

Opera and Chrome were identical in terms of the number and type of leaked addresses. This is likely because both are based on Google's open-source browser project, *Chromium*[8]. In all the individual tests that resulted in IP address leakage, they both revealed the temporary IPv6 address as well as either the local private IP address or the VPN-assigned private IP address. Somewhat different behaviour was exhibited by Firefox, which in most cases revealed either the local private IP address or the VPN-assigned private IP address; in some cases it did not reveal any addresses.

Firefox was the least privacy-damaging of the Windows-based browsers and Edge the most. In macOS, Safari revealed no IP addresses and so it is the least privacy-damaging. Chrome and Opera were the most privacy-damaging macOS browsers.

5.7 Countermeasures

The main lesson from the experiments described in this study is that users concerned about IP address leaks should select their browser and VPN service with care, perhaps using the `https://fingerprintable.org/webcrtleaks` site to check the properties of the chosen combination. Over and above this, users interested in maintaining their privacy by preventing WebRTC leaks can perform one or more of the countermeasures discussed in Chapter 7.

It is worth noting that disabling JavaScript or WebRTC itself would prevent WebRTC leaks, but would also disable many features and functionality of modern websites. Most users are likely to find this an unacceptably high cost for the privacy benefit they would receive, in the same way that whilst disabling cookies has significant privacy benefits, the usability impact is too great to make it a widely used protection measure.

5.8 Disclosure

We contacted all five of the VPN service providers tested in this study and provided them with our results. We were only received a response from ExpressVPN, who informed

[6] Still true as of April 2019.
[7] `https://ortc.org/faq` [accessed on 17/4/2019]
[8] `http://www.chromium.org/blink/developer-faq` [accessed on 14/05/2017]

us that they were working on resolving the problem. They subsequently provided us with the beta version of an upgraded client which, when tested, no longer leaked IP addresses.

5.9 Summary

We performed experiments with the five most widely used WebRTC-enabled browsers, i.e. Chrome, Firefox, Opera, Edge and Safari. We tested each of them with five widely used commercial VPN services in order to discover which client IP addresses can be revealed. Our experiments employed a specially established website which downloaded a slightly modified version of publicly available JavaScript to the client under test. The script fetches IP addresses made available via the browser WebRTC functionality. In most cases, at least one of the client IP addresses was leaked. Edge was the most seriously affected by WebRTC leaks, whereas Safari leaked no addresses at all. Our experiments revealed that the number and type of leaked IP addresses are affected by the choice of browser as well as the VPN service and program settings. We concluded the chapter by proposing countermeasures that can be used to help mitigate this problem.

Chapter 6

FingerprintAlert Browser Extension

6.1 Introduction

This chapter contributes to the goal of increasing end-user awareness about browsing fingerprinting. We describe *FingerprintAlert*, a Chrome browser extension that uses a novel approach to detect (and optionally block) browser fingerprinting. We discuss several anti-fingerprinting browser extensions in 7.3.2.

Commonly used browsers such as Chrome, Edge and Internet Explorer include very little functionality to help mitigate fingerprinting, alert the user to its occurrence, or even provide information about it in user help documents. Therefore, in order to help resist and raise awareness of browser fingerprinting, we developed a browser extension that alerts users whenever a visited website attempts to fingerprint their browser; users can also opt to enable a fingerprinting blocking feature. The development of this extension was developed in parallel with performing the study described in Chapter 3. The extension uses the same set of 17 attribute values to detect browser fingerprinting as were used in that study.

The remainder of this chapter is structured as follows. Section 6.2 provides a brief overview of the FingerprintAlert extension. In Section 6.3, we discuss its blocking feature. Section 6.4 provides details of its operation. In Section 6.5 we review its strengths and shortcomings. Section 6.6 describes some of the challenges that might limit its effectiveness. We describe its potential role in improving the awareness of users about browser fingerprinting in Section 6.7. We conclude the chapter with a summary in Section 6.8.

6.2 Overview

As part of the research described in Chapter 3, we developed *FingerprintAlert*[1,2], a browser extension usable with both Windows and macOS. It was initially developed to work on Chrome; however we later migrated it to also work with Firefox.

Based on the preliminary crawling described in 3.5.3, the extension detects possible browser fingerprinting by looking for the transmission of the values of the same set of 17 attributes as described in Chapter 3. That is, FingerprintAlert checks for the presence of the values these attributes take for the browser and host platform on which the extension is being run. It is activated whenever a web page is loaded, and checks whether any of these 17 attribute values are being relayed back to a web server. To the author's knowledge, this method of fingerprinting detection has not previously proposed in the literature.

If the extension detects such activity, it displays an alert that includes both the sending and receiving URLs (see Figure 6.1 for an example alert message). The extension also provides a detailed report of detected activities, including data relayed and the corresponding destination(s) (see Figure 6.2 for an example). Finally, the extension offers a user-selectable option to automatically block detected fingerprinting attempts. If selected, an HTTP message including any of the monitored attributes will be blocked from being relayed back to a remote server.

Figure 6.1: An example of a FingerprintAlert warning

6.3 Blocking

Websites typically send collected data in a series of HTTP messages, and if the blocking feature is activated *FingerprintAlert* blocks all HTTP messages that contain at least one of the 17 attribute values. This has a greater impact than just blocking the transfer

[1] https://chrome.google.com/webstore/detail/ielakmofegkdlpnlppfikmkbceajdofo
[2] https://addons.mozilla.org/en/firefox/addon/fingerprintalert

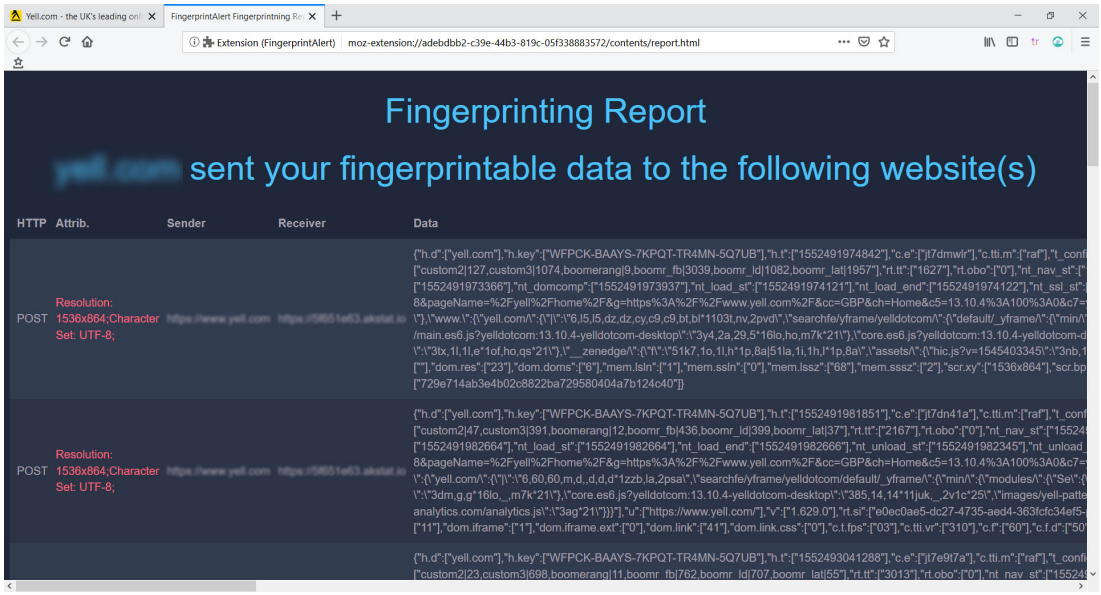

Figure 6.2: Example report produced by FingerprintAlert

of the 17 selected attribute values, since the experiments reported in Chapter 3 suggest that these attributes are typically transmitted in the same HTTP message as a large number of other fingerprinting attributes, which are also blocked as a result. We verified that the extension successfully blocked the messages by setting up an HTTP proxy server (Burp Suite Scanner[3]) that allowed us to monitor raw traffic between the browser and destination web servers.

As with any extension that interferes with browser behaviour, the blocking feature of *FingerprintAlert* might cause unexpected results or even break some websites. To ensure it does not cause significant usability issues, we tested it on the 50 most visited websites from the list used in the Chapter 3 experiments. We enabled the blocking feature, and spent around two minutes on each website performing actions such as signing up, logging in and clicking on links. During the tests we did not observe any unexpected behaviour or errors except for minor glitches on two websites (e.g. unable to load support chat window). Nonetheless, in the unlikely event that the extension damages a user's experience at a website, the blocking option or the notifications option can easily be disabled. The extension will continue to record detected fingerprinting attempts even if both blocking and notifications are disabled.

[3]This is a tool for testing web application security. https://portswigger.net/burp

6.4 Details of Operation

6.4.1 Overview

The extension is made up of two main components. The first component (see 6.4.3) consists of three pieces of JavaScript code that implement the functionality of the extension. The second component (see 6.4.2) is the interface, that is rendered using HTML, CSS and images. The user interface component incorporates three HTML files,

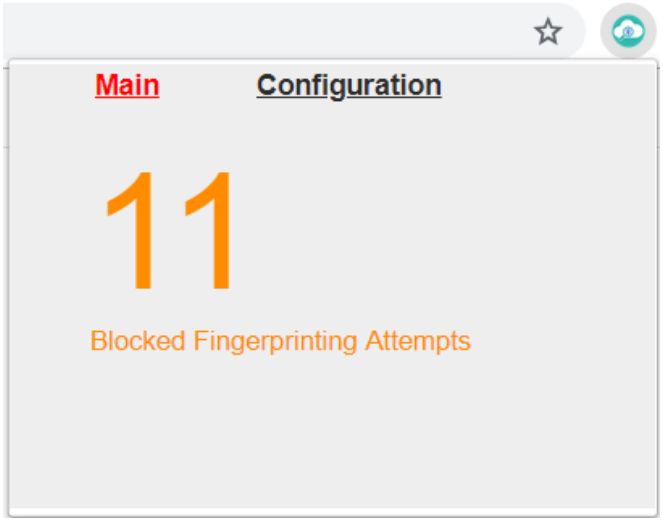

Figure 6.3: A partial browser screenshot showing the `Main` pop-up user interface

the pop-up user interface (see Figures 6.3 and 6.4), the fingerprinting detection report (see Figure 6.2) and the help page.

6.4.2 User Interface Component

The pop-up user interface consists of two pages, between which a user can switch by clicking on one of the top tabs. The first tab (i.e. `Main` — see Figure 6.3) displays the number of blocked fingerprinting attempts. The second tab (i.e. `Configuration` — see Figure 6.4) contains the following five options.

- The `Notifications` option switch (top left in Figure 6.4) is enabled by default and causes the extension to display in-browser alerts whenever a fingerprinting attempt is detected. The user can slide the switch to the left to disable these alerts.
- The `Block Fingerprinting` option switch (top right in Figure 6.4) is disabled by default and, if enabled, causes the extension to block detected fingerprinting

attempts. The user can slide the switch to the right to enable it.

- The `Fingerprinting Report` button (centre in Figure 6.4) opens a new page that contains details of detected fingerprinting attempts.
- The `Reset` button (bottom left in Figure 6.4) deletes all data collected by the extension and restores all options to their defaults.
- The `Help` button (bottom right in Figure 6.4) opens a page that serves as the extension's user manual.

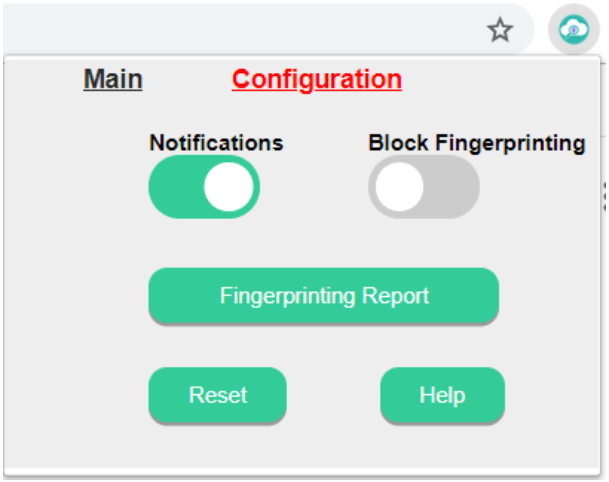

Figure 6.4: Partial browser screenshot showing the `Configuration` user interface

6.4.3 Functional Component

The three scripts making up the functional component are listed below along with a high-level description of their functions.

- `background.js` is the main script. It extracts the 17 browser attribute values for the client, and then monitors traffic between the client browser and remote web servers for the presence of any of these values. If the blocking option (see top right of Figure 6.4) is enabled, the script prevents the browser transmitting any HTTP message that contains one or more of of the attribute values.
- `popup.js` executes the user commands entered via the pop-up user interface.
- `buildmap.js` constructs reports detailing detected fingerprinting attempts.

The FingerprintAlert scripts can be found in the GitHub open repositories[4].

[4] https://github.com/fingerprintable/FingerprintAlert

6.4.4 Installation and Use

There are two versions of FingerprintAlert, one compatible with Chrome and the other with Firefox. They can be downloaded and installed by visiting the corresponding Chrome and Firefox official browser extension page[5,6].

After installation, the extension will operate immediately. Every time the browser is opened (or the extension re-installed), the extension will extract the values of the 17 browser attributes. Examples of extracted attribute values can be found in Table 3.4. It will then monitor all information transmitted between the client browser and any web server, looking for exact string matches with the 17 attribute values.

The extension monitors all outgoing traffic from the browser to a web server. If the notifications option is enabled, it will notify the user via an alert message (see Figure 6.1) whenever an HTTP message is discovered to contain one or more of the 17 attribute values. In parallel, if the blocking option is enabled, the transmission of any messages found to contain one or more of the 17 values will be blocked; the two options work independently. The extension will collect data for its reporting functions regardless of the options settings.

6.5 Review

6.5.1 Strengths

Despite the fact that *FingerprintAlert* only detects 17 attributes, these attributes are typically transmitted alongside other attributes which are also detected/blocked, given that they are in the same HTTP message. The extension can be used in conjunction with other anti-fingerprinting browser extensions (see 7.3.2) since it uses a different approach to detection and blocking.

6.5.2 Shortcomings

Of course, if a fingerprinting website does not collect any of the 17 attributes that *FingerprintAlert* searches for, then it will give a false negative. Similarly, if the website encodes the collected information in some way, i.e. so that a simple string search will fail, then again a false negative will result.

In its default setting, i.e. with blocking disabled, *FingerprintAlert* has no effect on the appearance and functionality of websites. However, as discussed in 6.3, if blocking is enabled then it may occasionally cause minor problems.

[5] https://chrome.google.com/webstore/detail/ielakmofegkdlpnlppfikmkbceajdofo
[6] https://addons.mozilla.org/en/firefox/addon/fingerprintalert

6.6 Challenges

Detecting novel approaches to fingerprinting is an obstacle that faces all privacy extensions (see 7.2.2). A major challenge facing the approach adopted by *FingerprintAlert* is that websites could choose to conceal transmitted attributes, e.g. using encryption, or use attributes that are not publicly known. Additionally, it is difficult to automatically detect all fingerprinting attribute values, as they may be similar to other data or have no specific set of values. On the other hand, the script analysis approach adopted widely by other anti-fingerprinting extensions, has its own issues. In particular detecting and examining scripts executed on websites is likely to be hindered by changes in code, syntax and execution.

6.7 Awareness

The main purpose of FingerprintAlert is to make users aware of fingerprinting attempts as they happen and the identity of domains collecting the fingerprinting data, and as a result increase their awareness of how widespread such practices are. The results of our study could also help in developing new tools designed to thwart fingerprinting.

If user privacy is to be preserved on the web, it is also important that fingerprinting technology is made user-controllable, so that users can limit the degree to which they are tracked. FingerprintAlert helps contribute to this by providing users with the option to block browser fingerprinting.

6.8 Summary

In this chapter we described *FingerprintAlert*, a freely available browser extension that detects fingerprinting attempts by visited websites. It also blocks detected attempts if the user enabled the option. Furthermore, it maintains a user-accessible record of detected fingerprinting attempts. This record includes the visited website, the information collected, the recipient web domain for the collected information and the HTTP method used. The extension is designed to make browser users aware of the prevalence of browser fingerprinting and to give them some control over it.

Chapter 7

Controlling Browser Fingerprinting

7.1 Introduction

In this chapter we focus on means of controlling browser fingerprinting. We look at measures that can be taken by both end-users and browser vendors. We examine previously proposed guidelines for browser vendors and formulate a comprehensive set of recommendations for browser vendors to give users greater control over browser fingerprinting. In a slightly different direction, we also propose a browser identifier that could serve as a standardized, and controllable, alternative to browser fingerprinting.

The fact that tracking can so readily be performed using browser fingerprinting is potentially a major threat to the privacy of web users, and as noted above it is one over which users currently have no control. Whilst there are uses of browser fingerprinting not directly relating to tracking, the lack of user control combined with the serious privacy threat suggests that means of limiting its effectiveness, i.e. what we refer to here as *fingerprinting countermeasures*, are of potentially huge importance, motivating the work described in this chapter.

Fingerprinting countermeasures can be divided into two categories, depending on whether they are directly implementable by the user regardless of the browser, or whether they require support from the browser vendor. In the remainder of this chapter we provide a comprehensive and systematic review of possible fingerprinting countermeasures. This is significant for a number of reasons. First, some of these techniques, whilst apparently known, have not previously been described in the academic literature. Second, this review enables us to compare their effectiveness (and their limitations) and also consider how best such countermeasures could be implemented,

from the perspectives of both the user and the browser vendor. Third, it enables us to identify areas where further research is urgently needed.

Finally, given that fingerprint-based tracking is so privacy intrusive (and uncontrollable), we also consider a way in which the major fingerprinters might be encouraged to abandon the practice. We, possibly controversially, propose that browsers should support a new type of website-accessible identifier, referred to as a *Unique Browser Identifier*, which would enable a level of user-controllable tracking without involving the collection of other user and browser data. This could make browser vendors willing to change behaviour to make fingerprinting difficult, leading to its use becoming redundant and (potentially) prevented. The possible operation of this identifier, and its advantages and disadvantages, are discussed.

The remainder of the chapter is organized as follows. In Section 7.2, a general overview of approaches to limiting fingerprinting is provided. Sections 7.3 and 7.4 provide detailed descriptions of all known user-based and browser-based anti-fingerprinting measures, respectively. In Section 7.5, we discuss a browser identifier-based proposal that aims at making browser fingerprinting redundant. Building on the previous sections, in Section 7.6 we review the degree to which browser fingerprinting can be controlled using current technology and consider ways in which greater control can be exercised in the future. The summary in Section 7.7 concludes the chapter.

7.2 Limiting Browser Fingerprinting

7.2.1 General Approaches

Most techniques aimed at limiting the effectiveness of fingerprinting either involve user-enabled options such as installing extensions, or operate via browsers that incorporate anti-fingerprinting features. We discuss these two general classes of countermeasures in greater detail in Sections 7.3 and 7.4 below.

A number of authors have proposed extensions that could help counter fingerprinting. Examples include *FP-Block* [80], *Blink* [43] and *FingerprintAlert* (see Chapter 6). There are also widely discussed extensions of this type that do not seem to have any corresponding published research, such as *CanvasBlocker*[1] and *Stop Fingerprinting*[2]. Luangmaneerote et al. [49] surveyed several of the more widely discussed fingerprinting countermeasures. They concluded that no single countermeasure can protect against all known fingerprinting methods, and that such countermeasures are likely to both negatively affect the user experience, and fail to block newly deployed fingerprinting

[1] https://addons.mozilla.org/en/firefox/addon/canvasblocker/
[2] https://addons.mozilla.org/en/firefox/addon/stop-fingerprinting

methods. In this chapter, we do not attempt to evaluate or enumerate individual anti-fingerprinting extensions; instead our goal is to consider the possible general approaches and in each case examine its effectiveness.

As far as browser-incorporated anti-fingerprinting techniques are concerned, a number of documents provide recommendations and guidelines to browser vendors aimed at limiting the effectiveness of fingerprinting, including RFC 6973 [13], a W3C Note [17], Eckersley [18], and Nikiforakis et al. [61]. However, the detailed technical aspects of such recommendations are outside the scope of this study, which is intended to provide a roadmap for policymakers, developers, browser vendors and interested general web users who wish to help control browser fingerprinting.

7.2.2 Challenges

A number of proposed browser fingerprinting countermeasures involve adding features to the browser, typically in the form of a browser extension. However, approaches of this type have serious limitations [49] and might even lead to effects opposite to those intended (see Section 7.3.3). By contrast, and as discussed in greater detail in Section 7.4, browser vendors could potentially control fingerprinting very effectively by making modifications to the way browsers operate. To help substantiate these claims, we next consider some of the main challenges in controlling browser fingerprinting.

Misuse of Browser Features

One major challenge in controlling fingerprinting is that information useful for fingerprinting can be obtained from "regular" web interactions, including website-originated browser scripts that access standardized APIs. That is, preventing fingerprinting might necessitate stopping, or restricting, such interactions, scripts and the APIs they access, which will almost certainly damage the user's browsing experience. For example, *canvas fingerprinting* makes use of the Canvas API (see Chapter 2), and simply blocking this API would result in browsers being unable to render images that could be critical to use of a web page.

Detection

Unlike tracking via cookies, that can be detected by the presence of cookies stored on user device, there are no unambiguous methods of detecting browser fingerprinting. Moreover, it can be performed passively (see Chapter 2) and is thus virtually undetectable (except, perhaps, by second order effects, such as receipt of targeted advertising). As discussed in Chapter 2, active fingerprinting can also be very hard to detect, as it can take advantage

of almost any existing web API. The same applies almost certainly to any future new APIs, unless new APIs are developed with built-in resistance to fingerprinting.

7.3 User-based Countermeasures

We now describe and analyze a variety of ways in which users can reduce the effectiveness of browser fingerprinting. We also consider the main challenges to user-based approaches, in particular observing that none of these techniques prevent fingerprinting — indeed, some may even make it more effective.

7.3.1 Browser Choice and Configuration

Browsers vary in their susceptibility to fingerprinting — see Chapter 4. These variations arise for a variety of reasons, including that some browsers, such as Firefox, have (possibly user-selectable) features that are designed to help resist fingerprinting. Moreover, as described by Boda et al. [10], some browser configuration options, such as disabling JavaScript, can affect fingerprinting. That is, a user's choice of browser and configuration options can change the effectiveness of fingerprinting. Finally, selecting a browser and version that is used by many users is also likely to help in making the browser a little less fingerprintable[3].

Before proceeding it is important to mention the Tor browser[4], which is specifically designed to be privacy-protecting; hence selecting Tor could be seen as a potentially effective user-based countermeasure. Unlike several widely-used browsers, the Tor browser includes by default a range of features intended to counter fingerprinting. These include using a fixed set of system colours, disabling plugins by default, limiting the number of fonts a document is allowed to load, and disallowing read access to canvas-rendered images unless express permission is granted by the user[5].

However, use of the Tor browser has a number of serious practical drawbacks including a seriously compromised browsing experience, one aspect of which is a slow browsing speed; there are also relatively few users (less than 1% of web users employ the Tor browser[6]) which might itself make fingerprinting possible because of the fingerprintability paradox (see 7.3.3). Tor also possesses other serious usability issues: it breaks some websites, some websites opt to block Tor clients, and changes in IP address negatively

[3] https://panopticlick.eff.org/self-defense

[4] The Tor browser is a modified version of Firefox that has enhanced security and privacy features, https://www.torproject.org/projects/torbrowser

[5] https://www.torproject.org/projects/torbrowser/design [accessed on 13/02/2019].

[6] Estimated by comparing the number of Tor browser clients during January 2019 as reported on https://metrics.torproject.org with the total number of web users reported on http://www.internetlivestats.com/internet-users.

affect the localized web browsing experience. Another example of a usability issue arises when Tor advises a user against maximizing the browser window via an in-browser notification. This is intended to keep the window at the default size and thereby the same size as other Tor users, hence preventing websites from learning the device's display size. Whilst this reduces browser fingerprintability, it will clearly have a negative effect on the user experience.

More generally, making attributes the same across multiple browser instances and hence reducing their uniqueness would make fingerprinting more difficult. However, it is also likely to damage the user experience by preventing a website tailoring its site to match the characteristics of the user device [66]. These serious disadvantages mean that the Tor browser is unlikely ever to be widely adopted, and hence cannot be seen as a generally applicable means of controlling fingerprinting.

7.3.2 Browser Extensions

Apart from the fundamental choice of browser, the main option for users wishing to limit the effectiveness of browser fingerprinting is to install one or more special-purpose browser extensions. As noted in Section 7.2.1, a number of such extensions exist, and we now consider these extensions in greater detail.

We were unable to find any detailed and generally applicable evidence regarding the relative effectiveness of the existing extensions; however they all seem to share common weaknesses. The limited effectiveness of some individual extensions has been evaluated in controlled environments, including by Nikiforakis et al. [62] and Luangmaneerote et al. [49]. It seems unlikely that a single extension is able to completely prevent fingerprinting, given the multiplicity of fingerprinting approaches, potentially including methods not in the public domain. Furthermore, there is no known method of learning whether extensions that counter certain fingerprinting techniques are in fact able to prevent real-world fingerprinting. This is especially apparent given that it is not always possible to detect when fingerprinting is occurring in the first place [1].

We also observe that some extensions not specifically designed for the purpose can nonetheless reduce the effectiveness of fingerprinting, as a by-product of their intended functionality. One such example is NoScript[7], that controls which scripts deployed by a visited website are allowed to run on the browser; depending on its configuration it might prevent execution of scripts used for browser fingerprinting. However, in this study we only consider purpose-built anti-fingerprinting extensions.

We next briefly review the three main techniques employed by the anti-fingerprinting browser extensions of which we are aware. We consider the limitations they all share in

[7] https://noscript.net

Section 7.3.3 below.

- **Script Blocking**: this works by blocking suspected fingerprinting scripts. Scripts are identified as suspect either via a blacklist of script providers and/or by their inclusion of known fingerprinting code. One example of an extension adopting the script blocking approach is *Privacy Badger*[8], developed by the Electronic Frontier Foundation. It detects canvas fingerprinting and prevents third-party scripts that deploy it from executing.

- **Attribute Spoofing**: extensions that use this technique attempt to prevent fingerprinting by constantly spoofing browser/platform attributes. Examples of extensions using this approach include: *PriVaricator* (due to Nikiforakis et al. [60]), *FPRrandom* (due to Laperdrix et al. [42]), *FPGuard* (due to FaizKhademi et al. [22]) and an unnamed extension due to Fiore et al. [27]. *FP-Block* (due to Torres et al. [80]), also fabricates some browser attributes, but it also blocks some scripts. That is, it employs both *script blocking* and *attribute spoofing*.

- **Data Blocking**: this involves blocking the retrieval of attributes that might be used for fingerprinting from the browser. This approach is used by *FingerprintAlert* (see Chapter 6).

7.3.3 Limitations

The main disadvantage of user-based countermeasures is that they all depend on manipulating or blocking data sent by the browser to remote web servers. This gives rise to two major limitations, which we now discuss.

- **Compromised Browsing Experience**: many browser-based countermeasures compromise the browsing experience in some way [49]. This is especially true when such countermeasures block certain scripts or spoof properties that may be important for the functionality of a visited website. There is also the possibility of a false positive, i.e. an incorrectly detected fingerprinting attempt, breaking some "innocent" websites.

- **Fingerprintability Paradox**: this phenomenon has been widely discussed — see, for example, Eckersley [18], Torres et al. [80], and Gulyás and Somé [31]. The term captures the fact that measures taken to reduce browser fingerprintability can unintentionally create a new source of fingerprinting. A simple example arises where the installation of an anti-fingerprinting browser extension that is only

[8] https://www.eff.org/privacybadger

installed in a small number of devices can be detected by a web server. That is, the presence of the extension is itself an attribute that can contribute to fingerprinting. This is a special case of what to refer to as *detection of defence*, where the deployment of a countermeasure can be detected and can be used to contribute to fingerprinting [61, 77]. This problem is especially significant if the number of users of the countermeasure is relatively small.

A related but distinct issue arises from the deployment of a browser extension that spoofs browser attributes for anonymization purposes but as a result exhibits a set of browser characteristics that is unrealistic or rare. This behaviour can make a browser more identifiable. Further, Vastel et al. [85] argue that some countermeasures potentially make browsers more fingerprintable since both spoofed and correct browser attributes can be discovered, e.g. using two different APIs. Finally, it has also been observed by Perry [65] that privacy-cautious users who opt to enable the *Do Not Track* (DNT)[9] option in their browsers increase their fingerprintability surface by doing so. In fact, the DNT option is no longer officially endorsed by the W3C [75]; while it is still supported by some browsers, Apple has announced[10] that version 12.1 of its Safari browser will drop support.

7.4 Browser-based Countermeasures

We next discuss the various countermeasures that could be implemented by browser vendors, as well as the associated challenges.

7.4.1 Reducing the Fingerprinting Surface

Having browsers exhibit similar information wherever possible would help limit fingerprinting [13, 17]. For example, as discussed in 2.3.1, the HTTP user agent header field is an important source of information for browser fingerprinting since, as currently implemented, it contains many browser and platform details, including full details of the browser version and, in some mobile browsers, the mobile phone model. Restricting the information included to only what is vital for the functioning of websites would clearly help reduce its utility for fingerprinting, whilst not affecting its usefulness for tailoring website content.

Currently, the specifics of API implementation are typically left to browser vendors,

[9] DNT is a standard browser feature that sends a request to websites that the browser user does not wish to be tracked [75].

[10] https://developer.apple.com/documentation/safari_release_notes/safari_12_1_release_notes [accessed on 13/02/2019]

which increases their usefulness for fingerprinting by enabling one browser to be distinguished from another through minor implementation differences [58]. However, if the standards included enough details to ensure these APIs are implemented in a way that would make browsers indistinguishable, then this will in turn limit fingerprinting effectiveness.

One of the methods used by some browsers (e.g. Firefox) to limit fingerprinting is attribute spoofing (as used in anti-fingerprinting extensions, cf. 7.3.2). However, as discussed in Section 7.3, attribute spoofing can seriously damage the user experience, and may also have a computational cost. Thus, whilst attribute spoofing may be necessary as a short-term expedient, in the longer term arranging for as much cross-browser attribute uniformity as possible is clearly a preferable approach [66]. As a result, it is likely that browsers that resort to spoofing are doing so as a temporary measure, as achieving behavioural uniformity would require, a currently absent, consensus amongst browser vendors.

In conclusion, and as recommended by RFC 6973 [13], it would be highly desirable for browsers to minimize the information content of browser-retrievable attributes to that needed to deliver the user experience, whilst also working to remove unnecessary differences in browser behaviour, including the order and presence of HTTP fields. Such changes could make a significant difference to the effectiveness of fingerprinting, with no obvious disadvantages in terms of the delivery of web content.

7.4.2 Context-based API Access Control

If browsers could be designed to control access to certain APIs based on the context of use, this would be very useful in controlling fingerprinting. For example, and as demonstrated by the *Fingerprintability* test web page (see 5.5), the client platform private IP address can be exposed via the WebRTC API. The specific feature of WebRTC that reveals the IP address is intended for video chat purposes. Currently, again as shown by *Fingerprintability*, this API can be accessed in many widely-used browsers regardless of whether or not video conferencing is taking place (or even without prompting the user). Hence (by some means) restricting access to the WebRTC API to video chat sites would clearly be beneficial in limiting fingerprinting.

There are many other examples of APIs whose use could be limited with similar benefits, such as access to processing and graphics hardware information through the WebGL API[11] in cases where it is not used to render any graphics. Analogously, since tracking is mostly performed by third parties, it would be very helpful if third-party

[11] A web API that renders 2D and 3D graphics on supported browsers.

scripts were prevented from accessing any API unless there is a clear reason for its use by a party other than the visited website [66].

7.4.3 Deprecate/Limit Unnecessary APIs

There is a need for the set of standardized APIs to be revisited and pruned where possible, since some APIs are apparently used almost exclusively by fingerprinters [66]. Of course, such APIs were not developed for fingerprinting purposes. Supporting this, Snyder et al. [76] have shown that many APIs are not utilized by any of top 100,000 visited websites. Such lightly used APIs create avoidable fingerprinting opportunities. Removing such APIs, or at least limiting their functionality, would therefore limit fingerprinting with minimal impact on the user experience. Two examples of such APIs are as follows.

- As noted by Olejnik et al. [64], the battery API is almost solely used for fingerprinting purposes. This API allows websites to detect the battery level of the client's device and optionally adjust website content to reduce battery-draining features if low battery levels are detected [41]. Safari never supported this API while Firefox did support it for a while but deprecated it in 2017 (possibly because of privacy issues). This move was followed by several other browser vendors. At the time of writing, Chrome is the only one of the top five browsers that still supports this API.

- An example of an API with features which seem primarily useful for fingerprinting is provided by the Canvas API, that enables a website to specify an image in code form (reducing data transfer requirements). Mowery and Shacham [56] showed that browsers and platforms vary in how they render canvas images. This is made valuable for fingerprinting by an API feature that allows a website to retrieve the canvas-rendered image. Disabling this latter feature would remove the fingerprinting function without preventing use of the API for its intended purpose.

7.4.4 Alerts and Prompts

As noted by Doty [17] and Fette and Melnikov [13], it would enhance user control if users were made aware whenever a browser detects behaviour which suggests fingerprinting is being performed. In addition, it would also be helpful if users could be given the means to control such behaviour. Of course, as we have discussed above, reliably detecting fingerprinting is a hard problem. However, it might be possible to detect when a website

is collecting information which is apparently unrelated to the information being served to the user. This could, perhaps, involve the use of machine learning techniques.

Examples of possible controls that could be given to users when a website is detected collecting fingerprintable data include:

- **blocking** the data collection;
- **anonymizing** the collectable data by spoofing or removing unique values;
- allowing users to **select** what data can be collected by a visited website.

Even if it is not possible to control the potential fingerprinting, it would help if users could be notified when such activity, e.g. involving the Canvas API, is detected. To a limited, and varying, extent this is implemented in both the Safari and Tor browsers. However, browser vendors need to be wary of *warning fatigue* [66], as over-frequent alerts might cause users to pay less attention to prompts and click on them without considering their content. Usefully, the number of times a user is prompted could be reduced if appropriate options were made available to users, such as enabling/disabling prompts, blacklisting, and automatically blocking certain third-party interactions. Moreover, prompts could be reduced if users were prompted only when suspicious behaviour is detected. An example of such suspicious behaviour would be a website that attempts to retrieve a canvas-rendered image while the image itself cannot be seen by the user because it is invisible or too small.

In a small investigation we found that none of the top four desktop browsers[12] (i.e. Chrome, Internet Explorer, Firefox and Edge) alert users when a visited website performs actions typical of fingerprinters, nor do they appear to incorporate any specific measures to prevent fingerprinting[13]. By contrast, as discussed in 7.3.1, the Tor browser protects against canvas fingerprinting by default as it prompts users before allowing the retrieval of canvas-rendered images by a visited website.

7.4.5 Reduction in API accuracy

Several authors (e.g. Eckersley [18] and Olejnik at al. [63]) have suggested using a reduction in the accuracy and level of detail provided by a browser in order to help counter fingerprinting. As discussed in 7.4.1, reducing the level of detail in the HTTP header would significantly help in limiting fingerprinting. The reporting of constantly

[12]The list of the most widely-used browsers was retrieved from https://www.netmarketshare.com/browser-market-share.aspx [accessed on 06/08/2018].

[13]Firefox has a small set of configurable anti-fingerprinting settings; however, these are only likely to be employed by technically aware users.

varying values, such as time and battery level could also be made less accurate to help prevent fingerprinting.

A further example of this type is provided by location information. Currently, all widely-used browsers prompt users to give permission if a website tries to access the location of the user. However, while the currently high level of accuracy obtainable (e.g. up to 4–5 metres) is likely to be necessary for applications such as satellite navigation, the need for such accuracy for most cases is arguable at best. Reducing the level of accuracy of the data provided, e.g. by enhancing the API to allow two or more levels of accuracy, could help limit fingerprinting. In addition to the prompt for user permission to access location information implemented by many browsers, the browser could also *warn* the user if a website is requesting highly-accurate location information.

7.4.6 Secure Data Handling

As described in 3.8.4, potentially privacy-sensitive fingerprinting data is often retrieved from a browser in plaintext via HTTP. Browsers could usefully enforce the use of HTTPS for such information transfers, as advised in RFC 6973 [13]. Currently, Chrome is the only browser that forces websites to use HTTPS in order to access the user location through the Geolocation API[14]. However, this restriction is not extended to other APIs.

Currently, several browsers such as Firefox warn users if they are visiting a website not using HTTPS, and prompt users to deny transmission if they are about to submit information that does not use it. However, no warnings and almost no restrictions are in place if a script from a third-party website retrieves information via HTTP. This shortcoming clearly merits consideration by browser vendors.

7.4.7 Challenges

Perhaps the main challenge for implementing browser-based countermeasures is to make browsers, wherever possible, behave indistinguishably to websites while keeping retrievable information to a minimum. Achieving the necessary agreement amongst competing vendor providers is likely to be difficult without enforcement by regulatory or standard bodies.

Another obvious challenge is ensuring changes implemented by browser vendors have minimal negative impact on the browsing experience of users. Achieving this is non-trivial given that, as discussed earlier, changes could include restricting APIs as well as occasionally prompting users.

[14] https://www.w3schools.com/html/html5_geolocation.asp [accessed on 18/02/2019]

Finally, some browser vendors might not wish to limit fingerprinting given that their parent companies apparently depend on it for their own services, such as providing web analytics and personalized online advertising.

7.5 Making Browser Fingerprinting Unnecessary?

7.5.1 A Different Approach

As noted briefly above, whilst browser vendors are in a strong position to decide how effective browser fingerprinting is, some of the key players in this space, notably Google, may be unlikely to take steps to limit it since, as shown by the study described in Chapter 3, they also appear to play a major role in browser fingerprinting. That is, there is clearly a desire by at least some key browser vendors to be able to track user behaviour. Indeed, to some extent this is necessary to enable these vendors to continue to support "free" (and highly valued) services, such as web search.

In this respect, it might be argued that trying to restrict cookies has been counter-productive for user privacy; at least cookies are, to a high degree, user-controllable, i.e. users can delete all cookies from time to time and thereby refresh their online identity. Exerting usage control is much more difficult when browser fingerprinting is employed for tracking, since as we have argued it is far less controllable and far more privacy-damaging in that it retrieves a wide variety of information about a user's platform and browser configuration. That is, pressure to limit cookies may have encouraged trackers to adopt browser fingerprinting, with an associated worsening of end-user privacy protection.

Apparently, five years ago Google intended to replace the use of cookies with some form of browser-generated ID (confirmed by a Google official [11]). However, no further information was ever made available. This apparently abandoned proposal suggests a possible new, and apparently paradoxical, approach to limiting browser fingerprinting. That is, if trackers can be offered a means of tracking browsers that is less privacy-damaging than browser fingerprinting, then browser vendors might be more willing to countenance adopting measures to reduce the effectiveness of fingerprinting.

Currently, operating systems such as Windows and Android support an *advertising ID* [74]. This serves as a unique identifier for the device/user for locally installed programs/apps to serve personalized ads. This ID can be reset, meaning that all previous associations are removed. This suggests a similar scheme in which a novel user-controllable ID is accessible by websites through browsers. This ID (which we call the *Unique Browser Identifier* (UBI)) would be managed by the browser itself, rather than the host operating system, since it is intended for use solely by the browser.

The main reason to introduce a browser-managed UBI is to provide a replacement

for both cookies and browser fingerprinting for the purposes of tracking, e.g. for the support of personalized advertising. That is, by providing a legitimate, simple and user-controllable method of enabling tracking, the use of cookies and browser fingerprinting for this purpose could be made redundant. Simultaneously with the introduction of the UBI, measures would also need to be put in place to prevent trackers using both the UBI and browser fingerprinting (hence damaging privacy even further). This could be achieved by regulation and/or standardization, as well as by implementing the recommendations given in Section 7.4. Apart from making fingerprinting for tracking redundant, the UBI could also replace other existing uses of browser fingerprinting. In particular it could serve as an additional layer of authentication.

7.5.2 Configuring Identifiers

To ensure that the introduction of the UBI gives the user the control that is currently lacking with browser fingerprinting, browsers will need to enable users to limit access to the UBI (and reset it). It could be advantageous for a browser to support more than one UBI, for example one for personalization (e.g. for personalized advertising) and another for authentication purposes (e.g. as a factor in multi-factor authentication). This would allow the access settings for the various UBIs to be separately configurable. For example, a user might choose to allow third parties to have access to a personalization UBI, e.g. as used for personalized advertising, and might also choose to automatically reset this UBI at fixed intervals (e.g. monthly). In parallel, an authentication UBI might be configured to only be available under very restricted circumstances, e.g. only to the site visited by the user and/or to a "login" web page and/or if the user gives explicit permission. UBIs, depending on their type, might also be unique per website, as opposed to being the same regardless of the retrieving party.

7.5.3 UBI and Cookies

It could be argued that the functionality of a UBI could just as easily be implemented through the use of a special cookie. However, as we discuss below, there are a number of reasons why the UBI offers desirable features not accessible through the simple use of cookies. Of course the UBI would not replace cookies, as they serve as a general-purpose means of adding state to HTTP, an otherwise stateless protocol.

One major difference between cookies and the proposed UBI is that the user has no means of controlling how cookies are used; all a user can do is have them deleted. By contrast, the UBI has a well-defined role and its use could be configured to meet user privacy requirements. That is, by providing explicit browser support for the UBI, its

use can be controlled much more precisely than would be the case for cookies.

To help enable transparency of use and make websites accountable for their actions, browsers could usefully maintain a log of accesses to each type of UBI, e.g. including access date/time and accessing URL, just as is the case for cookies. However, unlike the case for cookies, it would also be helpful to log details of whether the requesting site is a third-party site, while also identifying the first-party site whose website contained the third-party link. Recording this additional information is made possible by implementing the UBI as a distinct browser feature.

Browsers should also enforce the use of HTTPS for UBI transfers, i.e. preventing access via HTTP. Currently, unless the appropriate flag is set, cookies can be transferred via HTTP, and transferring the UBI unencrypted would clearly be a privacy risk. As we discussed in Chapter 3, it is interesting to observe that browser fingerprinting information is currently often transferred using HTTP.

Currently the expiry date of cookies is controlled by the server that created them, whereas UBI expiry would ideally be user-controllable. Moreover, if a multi-UBI system was implemented, UBIs would serve a range of purposes, and their behaviour and handling could be managed individually.

7.5.4 Privacy Considerations

Ideally, the use and functioning of a UBI should be standardized by an official body rather than being left to an initiative by a browser vendor. As mentioned in 7.5.1, Google considered replacing the use of cookies with a browser generated ID. Had such functionality been unilaterally added, perhaps with minimal user controls, it would potentially have given Google even greater control over online personalized advertising, to the detriment of user privacy, given that Google currently [19, 59] also owns the largest shares of both online advertising and browser users.

It might be argued that potential sharing of a UBI amongst trackers might increase privacy concerns. Analogously, concerns about sharing of browser fingerprinting IDs have recently been expressed [23]. However, in the case of UBIs, this might be avoided if the recommendations below and those in 7.5.1 and 7.5.3 are followed.

To help minimize the threat posed to user privacy, we propose that the following rules governing UBIs should be enforced by the browser.

- **UBI Access Control**: users should have full control over the use of all UBIs. This can be through prompts as well as enabling blacklisting/whitelisting of websites. Usage control should include providing means to enable/disable UBI access to third party websites. Denied websites could be provided with a dummy

ID to prevent them learning that the user has denied them access.

- **UBI Reset**: like the existing advertising IDs, UBIs should be resettable, allowing users to opt to be forgotten by all websites that possess any of the previous UBI(s).
- **UBI Opacity**: UBIs should be generated in a way that reveals no information about its platform/client, nor linking with previously generated values.

7.6 Discussion

7.6.1 Browsers with Fingerprinting-Resisting Features

Some initial steps have been taken by certain browser vendors to tackle browser fingerprinting. However, apart from the Tor browser, these steps remain small and many browser vendors have not added any such features to their browsers. Firefox calls its techniques of this type `resistFingerprinting`. Using the term "resist" seems appropriate given that the measures certainly do not prevent it altogether. Firefox version 62 incorporates four options that can be enabled to resist fingerprinting, although some are disabled by default.

It is not clear what exactly they do, but apparently one of them reduces time-reporting precision and another tackles one aspect of canvas fingerprinting. For reasons that are unclear, Firefox has opted not to make these options available in the main *Options* menu; instead, access is via the somewhat obscure advanced options, that are only accessible by typing `about:config` in the address bar (see Figure 7.1).

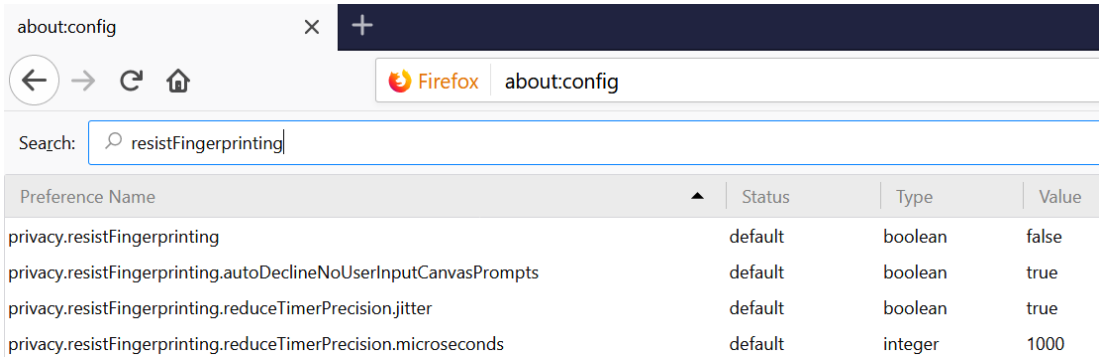

Figure 7.1: Firefox anti-fingerprinting options

Apart from these limited measures in Firefox, only Safari and the Tor browser appear to contain significant anti-fingerprinting features. Even in this case, only the latest Safari version possesses such features. This leaves around 98%[15] of web users without

[15] https://netmarketshare.com/browser-market-share.aspx [accessed 30/10/2018].

any default anti-fingerprinting protection. The Safari and Tor browser implementations of anti-fingerprinting measures differ, although both Safari and Tor are designed to make all versions indistinguishable. It is important that other browsers follow similar design considerations.

7.6.2 A Possible Role for Regulation

As discussed in 7.4, a number of steps could be taken by browser vendors to make fingerprinting significantly less effective, including removing (or limiting) rarely used APIs and reducing API accuracy. The main question is how to persuade browser vendors to implement such steps. One possible route might involve some kind of regulation.

More than 92%[16] of users use one of the five most widely used browsers, but trackers and the domains they use are very numerous. It therefore seems reasonable to focus more on controlling browser fingerprinting by regulating browsers rather than by regulating websites. Moreover, browsers act as a kind of middle man between websites and users, and can thus act as a type of privacy regulator [58]. However, it is important to note that some browser vendors also make extensive use of browser fingerprinting, and are thus likely to be reluctant to restrict it. Finally, despite the primary focus on browsers, the regulation of websites remains an important possibility, especially given that it appears that most tracking is performed by a limited number of third parties (see Chapter 3).

7.7 Summary

Browser fingerprinting is increasingly being used for online tracking of users, and, unlike the use of cookies, is almost impossible for users to control. This has a major negative impact on online privacy. Despite the availability of a range of fingerprinting countermeasures as well as some limited attempts by browser vendors to curb its effectiveness, it remains largely uncontrolled. Third-party countermeasures have inherent limitations and many browser vendors do not appear to have made significant efforts to control it. This chapter provided a comprehensive and structured discussion of measures to limit or control browser fingerprinting, covering both user-based and browser-based techniques. It also discusses the limitations of these measures and the need for browser vendor support in controlling fingerprinting. Further, a somewhat counterintuitive possible new browser identifier is proposed which could make cookies and fingerprint-based tracking redundant; the need for, and possible effect of, this feature had been discussed.

[16] https://netmarketshare.com/browser-market-share.aspx [accessed 30/08/2018].

Chapter 8

Conclusions and Future Work

8.1 Conclusions

The work described in this thesis has been motivated by the growing privacy threat posed by browser fingerprinting. Of course, addressing such a threat requires that we understand in detail how it works and who is doing it. Chapters 2 to 5 of this thesis describe work directed at learning more about the prevalence and nature of browser fingerprinting. The rest of the thesis builds on the understanding derived from the first chapters, and considers the question of giving users greater control over browser fingerprinting. We next examine in greater detail the contributions in each of the chapters of the thesis.

In Chapter 2 we reviewed browser fingerprinting and its related prior art. We observed that, since users have no direct means of controlling it, in some ways browser fingerprinting poses a higher risk to privacy than tracking via cookies. We also observed that it is increasingly being used for online tracking of users even in the absence of a persistent IP address or cookie.

In Chapter 3 we reviewed prior art that attempted to measure the prevalence of browser fingerprinting. We explained how the variety of different fingerprinting detection methods used in previous studies has given rise significantly varying results. In particular, most previous studies have been based on detecting a very limited number of techniques or scripts. We also described the results of a study showing that browser fingerprinting is being conducted on a significantly larger scale than previously reported, involving the transmission of large volumes of browser and device-specific data to trackers. We also reported on the large number of fingerprinting attributes collected. As other authors have described, browser fingerprinting has significant negative implications on user privacy, and it is therefore important that the web user community is made aware of

its prevalence and potential effectiveness. To this end, we developed *FingerprintAlert*, used in the study and discussed in detail in Chapter 6.

In Chapter 4 we discussed an investigation of an aspect of browser fingerprinting that has not previously been explored in the literature, namely the differences between browsers in terms of the amount of information they reveal to executing scripts (and hence to fingerprinting websites). For example, some mobile browsers reveal (for no obvious reason) the specific phone model, and browsers differ widely in how they implement the WebRTC and Canvas APIs, both of key importance for browser fingerprinting. It would therefore be highly desirable if all browsers asked for user permission before rendering a canvas image, or at least disabled the option that allows servers to retrieve details of the rendered image. At the time we performed our experiments, Safari would appear to be the best choice in this respect on both mobile and desktop platforms. Despite Chrome being the most widely used browser, it proved to be one of the most fingerprintable.

In Chapter 5 we reported on experiments examining the disclosure of IP addresses via the WebRTC API. In this study, Safari did not allow any client IP addresses to be leaked via WebRTC. Edge, on the other hand, proved to be the most privacy-damaging in this respect. However, regardless of the user browser choice, we found that some VPN implementations prevent leaking of client public IP address(es). Moreover, in some cases, selecting an appropriate client VPN configuration fully or partially prevented disclosure of IP addresses via WebRTC. The experiments we performed in this study explored an aspect of WebRTC leaks that has not been addressed in previous work, namely that the choice of browser and VPN service can make a significant difference to the number of IP addresses that are leaked. The results will help users decide on best practices to minimize the risk of loss of IP confidentiality. We also hope it will encourage VPN and browser providers to work on mitigating the privacy-compromising properties of their implementations of the WebRTC API.

In Chapter 6, we described *FingerprintAlert*, a freely available browser extension that detects, and optionally blocks, transfers of data likely to be used for browser fingerprinting. It also provides users with detailed reports of detected fingerprinting. Whilst it is not 100% accurate, the main purpose of developing the extension was to make browser users aware of the wide use of browser fingerprinting and to enable them to exert some control over it.

In Chapter 7 we discussed how browser fingerprinting is becoming commonplace, with browsers leaving users unequipped with the means to control it. This is likely to damage online privacy. This privacy risk is particularly serious since third-party countermeasures are inherently limited. In the chapter we reviewed and analysed a

wide range of measures that can be performed by browser vendors to help address this problem. We also propose a new browser identifier that would serve as a standardized and controllable method for online tracking.

8.2 Future Work

In Chapter 3 we described a study aimed at learning how many of the 10,000 most visited websites are performing browser fingerprinting. In this study, detection was based on identifying the presence of one or more of 17 specific attribute values in HTTP messages sent by the browser. No doubt it would be useful for future studies of this type to include a larger number of attributes. Moreover, increasing the number of the crawled websites would also be valuable; indeed ideally we should try to examine all the active websites on the web.

As discussed in 4.2.4, the unique *device IDs* that are assigned to video/audio devices (e.g. microphones or loudspeakers), and that are revealed by the WebRTC API, are potentially a rich source of information for fingerprinting. This is particularly so if they remain unchanged for long periods of time, as is the case for some browsers. Despite the privacy risk they pose, to the author's knowledge, no study has evaluated their potential role in fingerprinting. Moreover, the privacy risk posed by these IDs would be increased if the current ID for a device could somehow be linked to previous IDs. Clearly, further studies are needed to better understand these device IDs, their potential usefulness in fingerprinting, and the reason for the varying implementations by browsers.

In the near future we aim to improve *FingerprintAlert* by increasing the number of automatically-detectable attributes. This can be achieved by further in-depth examination of the formats and values of attributes that are currently undetectable. Since the crawler used to perform the experiments described in Chapter 3 is based on the extension, any future crawls would also be made more effective by such improvements. For example, the device ID values discussed in Chapter 4 could be included as a new detectable attribute. However, the usefulness of including such an attribute depends on it remaining unchanged for long enough to be useful in fingerprinting and tracking.

If it is to be pursued further, the UBI proposed in 7.5 would require a detailed analysis of its possible implementation and use, as well as its impact on user privacy both from an ethical and a legal/regulatory perspective. To have any chance of it being adopted as a possible standard feature for inclusion in browsers, it would be necessary to get wider agreement that such an approach is desirable. That is, promoting an open debate on such an issue, e.g. through conference presentations and panel discussions, would appear to be the logical next step.

Bibliography

[1] G. Acar, C. Eubank, S. Englehardt, M. Juárez, A. Narayanan, and C. Díaz. The web never forgets: Persistent tracking mechanisms in the wild. In G. Ahn, M. Yung, and N. Li, editors, *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security, Scottsdale, AZ, USA, November 3–7, 2014*, pages 674–689. ACM, 2014. 1, 11, 16, 17, 19, 30, 68

[2] G. Acar, M. Juárez, N. Nikiforakis, C. Díaz, S. F. Gürses, F. Piessens, and B. Preneel. Fpdetective: dusting the web for fingerprinters. In A. Sadeghi, V. D. Gligor, and M. Yung, editors, *2013 ACM SIGSAC Conference on Computer and Communications Security, CCS'13, Berlin, Germany, November 4–8, 2013*, pages 1129–1140. ACM, 2013. 8, 9, 12, 14, 17, 18, 19, 21, 31

[3] N. M. Al-Fannah. One leak will sink a ship: WebRTC IP address leaks. In *International Carnahan Conference on Security Technology, ICCST 2017, Madrid, Spain, October 23–26, 2017*, pages 1–5. IEEE, 2017. 47

[4] N. M. Al-Fannah and W. Li. Not all browsers are created equal: Comparing web browser fingerprintability. In S. Obana and K. Chida, editors, *Advances in Information and Computer Security — 12th International Workshop on Security, IWSEC 2017, Hiroshima, Japan, August 30 – September 1, 2017, Proceedings*, volume 10418 of *Lecture Notes in Computer Science*, pages 105–120. Springer, 2017. 34

[5] N. M. Al-Fannah, W. Li, and C. J. Mitchell. Beyond cookie monster amnesia: Real world persistent online tracking. In L. Chen, M. Manulis, and S. Schneider, editors, *Information Security — 21st International Conference, ISC 2018, Guildford, UK, September 9–12, 2018, Proceedings*, volume 11060 of *Lecture Notes in Computer Science*, pages 481–501. Springer, 2018. 16

[6] F. Alaca and P. C. van Oorschot. Device fingerprinting for augmenting web authentication: classification and analysis of methods. In S. Schwab, W. K. Robertson,

and D. Balzarotti, editors, *Proceedings of the 32nd Annual Conference on Computer Security Applications, ACSAC 2016, Los Angeles, CA, USA, December 5–9, 2016*, pages 289–301. ACM, 2016. 7, 10, 13, 19, 24, 31, 42, 48, 49

[7] A. Barth. HTTP state management mechanism. RFC 6265, RFC Editor, April 2011. http://www.rfc-editor.org/rfc/rfc6265.txt. 7

[8] M. A. Bashir, S. Arshad, E. Kirda, W. K. Robertson, and C. Wilson. How tracking companies circumvented ad blockers using websockets. In *Proceedings of the Internet Measurement Conference 2018, IMC 2018, Boston, MA, USA, October 31 – November 02, 2018*, pages 471–477. ACM, 2018. 31

[9] A. Bergkvist, D. C. Burnett, C. Jennings, B. A. Anant Narayanan, T. Brandstetter, and J.-I. Bruaroey. WebRTC 1.0: Real-time communication between browsers. Candidate recommendation, W3C, September 2018. https://www.w3.org/TR/webrtc. 11

[10] K. Boda, Á. M. Földes, G. G. Gulyás, and S. Imre. User tracking on the web via cross-browser fingerprinting. In P. Laud, editor, *Information Security Technology for Applications — 16th Nordic Conference on Secure IT Systems, NordSec 2011, Tallinn, Estonia, October 26–28, 2011, Revised Selected Papers*, volume 7161 of *Lecture Notes in Computer Science*, pages 31–46. Springer, 2011. 7, 67

[11] Business Insider. Google And Facebook To Replace Cookies. https://www.businessinsider.com/google-and-facebook-to-replace-cookies-2014-2?r=US&IR=T, 2014. Accessed 19/12/2018. 75

[12] Y. Cao, S. Li, and E. Wijmans. (Cross-)browser fingerprinting via OS and hardware level features. In *24th Annual Network and Distributed System Security Symposium, NDSS 2017, San Diego, California, USA, February 26 – March 1, 2017*. The Internet Society, 2017. 8, 31, 37, 42

[13] A. Cooper, H. Tschofenig, B. Aboba, J. Peterson, J. Morris, M. Hansen, and R. Smith. Privacy considerations for internet protocols. RFC 6973, RFC Editor, July 2013. https://tools.ietf.org/rfc/rfc6973.txt. 66, 70, 71, 72, 74

[14] Council of the European Union. Regulation (EU) 2016/679 (General Data Protection Regulation), April 2016. https://publications.europa.eu/en/publication-detail/-/publication/3e485e15-11bd-11e6-ba9a-01aa75ed71a1. 1

[15] D. Crockford. *JavaScript: The Good Parts*. Yahoo Press, Sebastopol, California, 2008. 6

[16] A. Das, G. Acar, N. Borisov, and A. Pradeep. The web's sixth sense: A study of scripts accessing smartphone sensors. In D. Lie, M. Mannan, M. Backes, and X. Wang, editors, *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS 2018, Toronto, ON, Canada, October 15–19, 2018*, pages 1515–1532. ACM, 2018. 19, 20

[17] N. Doty. Fingerprinting guidance for web specification authors (draft). Interest group note, W3C, November 2015. https://www.w3.org/TR/fingerprinting-guidance. 2, 9, 66, 70, 72

[18] P. Eckersley. How unique is your web browser? In M. J. Atallah and N. J. Hopper, editors, *Privacy Enhancing Technologies, 10th International Symposium, PETS 2010, Berlin, Germany, July 21–23, 2010. Proceedings*, volume 6205 of *Lecture Notes in Computer Science*, pages 1–18. Springer, 2010. 1, 7, 8, 9, 10, 11, 12, 14, 16, 31, 38, 41, 66, 69, 73

[19] eMarketer Inc. Global ad spending update. https://www.emarketer.com/content/global-ad-spending-update, 2019. Accessed 25/02/2019. 77

[20] S. Englehardt and A. Narayanan. Online tracking: A 1-million-site measurement and analysis. In E. R. Weippl, S. Katzenbeisser, C. Kruegel, A. C. Myers, and S. Halevi, editors, *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, October 24–28, 2016*, pages 1388–1401. ACM, 2016. 9, 13, 18, 19, 21, 26, 29, 32, 39, 49

[21] European Commission. Cookies. http://ec.europa.eu/ipg/basics/legal/cookies/index_en.htm. Accessed 11/04/2019. 1

[22] A. FaizKhademi, M. Zulkernine, and K. Weldemariam. Fpguard: Detection and prevention of browser fingerprinting. In P. Samarati, editor, *Data and Applications Security and Privacy XXIX — 29th Annual IFIP WG 11.3 Working Conference, DBSec 2015, Fairfax, VA, USA, July 13–15, 2015, Proceedings*, volume 9149 of *Lecture Notes in Computer Science*, pages 293–308. Springer, 2015. 7, 18, 19, 20, 30, 69

[23] M. Falahrastegar, H. Haddadi, S. Uhlig, and R. Mortier. Tracking personal identifiers across the web. In T. Karagiannis and X. A. Dimitropoulos, editors, *Passive and Active Measurement — 17th International Conference, PAM 2016,*

Heraklion, Greece, March 31 – April 1, 2016. Proceedings, volume 9631 of *Lecture Notes in Computer Science*, pages 30–41. Springer, 2016. 32, 77

[24] I. Fette and A. Melnikov. The WebSocket protocol. RFC 6455, RFC Editor, December 2011. http://www.rfc-editor.org/rfc/rfc6455.txt. 31

[25] R. Fielding and J. Reschke. Hypertext transfer protocol (HTTP/1.1): Semantics and content. RFC 7231, RFC Editor, June 2014. http://www.rfc-editor.org/rfc/rfc7231.txt. 6, 9, 10, 13, 23

[26] D. Fifield and S. Egelman. Fingerprinting web users through font metrics. In R. Böhme and T. Okamoto, editors, *Financial Cryptography and Data Security — 19th International Conference, FC 2015, San Juan, Puerto Rico, January 26–30, 2015, Revised Selected Papers*, volume 8975 of *Lecture Notes in Computer Science*, pages 107–124. Springer, 2015. 7, 37

[27] U. Fiore, A. Castiglione, A. D. Santis, and F. Palmieri. Countering browser fingerprinting techniques: Constructing a fake profile with google chrome. In L. Barolli, F. Xhafa, M. Takizawa, T. Enokido, A. Castiglione, and A. D. Santis, editors, *17th International Conference on Network-Based Information Systems, NBiS 2014, Salerno, Italy, September 10–12, 2014*, pages 355–360. IEEE Computer Society, 2014. 7, 36, 69

[28] J. Franklin and D. McCoy. Passive data link layer 802.11 wireless device driver fingerprinting. In A. D. Keromytis, editor, *Proceedings of the 15th USENIX Security Symposium, Vancouver, BC, Canada, July 31 – August 4, 2006*. USENIX Association, 2006. 7

[29] A. Gómez-Boix, P. Laperdrix, and B. Baudry. Hiding in the crowd: an analysis of the effectiveness of browser fingerprinting at large scale. In P. Champin, F. L. Gandon, M. Lalmas, and P. G. Ipeirotis, editors, *Proceedings of the 2018 World Wide Web Conference on World Wide Web, WWW 2018, Lyon, France, April 23–27, 2018*, pages 309–318. ACM, 2018. 10

[30] D. Gorley and B. Totty. *HTTP: The Definitive Guide*. O'Reilly Media, Sebastopol, California, 2002. 6

[31] G. G. Gulyás, D. F. Somé, N. Bielova, and C. Castelluccia. To extend or not to extend: on the uniqueness of browser extensions and web logins. *CoRR*, abs/1808.07359, 2018. http://arxiv.org/abs/1808.07359. 69

[32] Y. Haga, Y. Takata, M. Akiyama, and T. Mori. Building a scalable web tracking detection system: Implementation and the empirical study. *IEICE Transactions*, 100-D(8):1663–1670, 2017. 18, 19, 20, 30

[33] M. Hazhirpasand and M. Ghafari. One leak is enough to expose them all — from a WebRTC IP leak to web-based network scanning. In M. Payer, A. Rashid, and J. M. Such, editors, *Engineering Secure Software and Systems — 10th International Symposium, ESSoS 2018, Paris, France, June 26–27, 2018, Proceedings*, volume 10953 of *Lecture Notes in Computer Science*, pages 61–76. Springer, 2018. 55

[34] R. Hinden and B. Haberman. Unique local IPv6 unicast addresses. RFC 4193, RFC Editor, October 2005. https://tools.ietf.org/rfc/rfc4193.txt. 48

[35] R. Hosoi, T. Saito, T. Ishikawa, D. Miyata, and Y. Chen. A browser scanner: Collecting intranet information. In *19th International Conference on Network-Based Information Systems, NBiS 2016, Ostrava, Czech Republic, September 7-9, 2016*, pages 140–145. IEEE Computer Society, 2016. 12, 49

[36] D. Jackson and J. Gilbert. WebGL 2.0 specification. Khronos working draft, Khronos Group, February 2019. https://www.khronos.org/registry/webgl/specs/latest/2.0. 13

[37] C. Jakobsson. Peer-to-peer communication in web browsers using WebRTC a detailed overview of WebRTC and what security and network concerns exists. Master's thesis, Umeå University, Department of Computing Science, 2015. http://www8.cs.umu.se/education/examina/Rapporter/ChristerJakobsson.pdf. 12, 49

[38] A. Keranen, C. Holmberg, and J. Rosenberg. Interactive connectivity establishment (ICE): A protocol for network address translator (NAT) traversal. RFC 8445, RFC Editor, July 2018. https://tools.ietf.org/rfc/rfc8445.txt. 12

[39] R. Koch, M. Golling, and G. D. Rodosek. Geolocation and verification of ip-addresses with specific focus on IPv6. In G. Wang, I. Ray, D. Feng, and M. Rajarajan, editors, *Cyberspace Safety and Security*, volume 8300 of *Lecture Notes in Computer Science*, pages 151–170. Springer International Publishing, 2013. 47

[40] T. Kohno, A. Broido, and K. C. Claffy. Remote physical device fingerprinting. In *2005 IEEE Symposium on Security and Privacy (S&P 2005), 8-11 May 2005, Oakland, CA, USA*, pages 211–225. IEEE Computer Society, 2005. 10

[41] M. Lamouri and A. Kostiainen. Battery status API. Candidate recommendation, W3C, July 2016. https://www.w3.org/TR/2016/CR-battery-status-20160707. 72

[42] P. Laperdrix, B. Baudry, and V. Mishra. FPRandom: Randomizing core browser objects to break advanced device fingerprinting techniques. In E. Bodden, M. Payer, and E. Athanasopoulos, editors, *Engineering Secure Software and Systems — 9th International Symposium, ESSoS 2017, Bonn, Germany, July 3-5, 2017, Proceedings*, volume 10379 of *Lecture Notes in Computer Science*, pages 97–114. Springer, 2017. 69

[43] P. Laperdrix, W. Rudametkin, and B. Baudry. Mitigating browser fingerprint tracking: Multi-level reconfiguration and diversification. In *10th IEEE/ACM International Symposium on Software Engineering for Adaptive and Self-Managing Systems, SEAMS 2015, Florence, Italy, May 18–19, 2015*, pages 98–108, 2015. 65

[44] P. Laperdrix, W. Rudametkin, and B. Baudry. Beauty and the beast: Diverting modern web browsers to build unique browser fingerprints. In *IEEE Symposium on Security and Privacy, SP 2016, San Jose, CA, USA, May 22–26, 2016*, pages 878–894. IEEE Computer Society, 2016. 7, 8, 11, 13, 14, 29, 37, 42, 46

[45] A. Lerner, A. K. Simpson, T. Kohno, and F. Roesner. Internet jones and the raiders of the lost trackers: An archaeological study of web tracking from 1996 to 2016. In T. Holz and S. Savage, editors, *25th USENIX Security Symposium, USENIX Security 16, Austin, TX, USA, August 10–12, 2016*. USENIX Association, 2016. 8, 13, 32

[46] T. Li, H. Hang, M. Faloutsos, and P. Efstathopoulos. Trackadvisor: Taking back browsing privacy from third-party trackers. In J. Mirkovic and Y. Liu, editors, *Passive and Active Measurement — 16th International Conference, PAM 2015, New York, NY, USA, March 19–20, 2015, Proceedings*, volume 8995 of *Lecture Notes in Computer Science*, pages 277–289. Springer, 2015.

[47] T. Libert. Exposing the invisible web: An analysis of third-party http requests on 1 million websites. *International Journal of Communication*, 9:18, October 2015. 1, 14, 18, 19, 20, 29, 32

[48] X. Liu, Q. Liu, X. Wang, and Z. Jia. Fingerprinting web browser for tracing anonymous web attackers. In *IEEE First International Conference on Data Science in Cyberspace, DSC 2016, Changsha, China, June 13-16, 2016*, pages 222–229. IEEE Computer Society, 2016. 49

[49] S. Luangmaneerote, E. Zaluska, and L. Carr. Survey of existing fingerprint countermeasures. In *2016 International Conference on Information Society (i-Society)*, pages 137–141. IEEE Computer Society, October 2016. 65, 66, 68, 69

[50] R. Mahy, P. Matthews, and J. Rosenberg. Traversal using relays around NAT (TURN): Relay extensions to session traversal utilities for NAT (STUN). RFC 5766, RFC Editor, April 2010. http://www.rfc-editor.org/rfc/rfc5766.txt. 12

[51] J. R. Mayer. *Any person... a pamphleteer: Internet Anonymity in the Age of Web 2.0.* Bachelor's thesis, Princeton University, 2009. 7

[52] J. R. Mayer and J. C. Mitchell. Third-party web tracking: Policy and technology. In *IEEE Symposium on Security and Privacy, SP 2012, 21–23 May 2012, San Francisco, California, USA*, pages 413–427. IEEE Computer Society, 2012. 1, 13, 14, 16

[53] G. Merzdovnik, M. Huber, D. Buhov, N. Nikiforakis, S. Neuner, M. Schmiedecker, and E. R. Weippl. Block me if you can: A large-scale study of tracker-blocking tools. In *2017 IEEE European Symposium on Security and Privacy, EuroS&P 2017, Paris, France, April 26-28, 2017*, pages 319–333. IEEE, 2017. 31

[54] H. Metwalley, S. Traverso, M. Mellia, S. Miskovic, and M. Baldi. The online tracking horde: A view from passive measurements. In M. Steiner, P. Barlet-Ros, and O. Bonaventure, editors, *Traffic Monitoring and Analysis — 7th International Workshop, TMA 2015, Barcelona, Spain, April 21–24, 2015. Proceedings*, volume 9053 of *Lecture Notes in Computer Science*, pages 111–125. Springer, 2015. 27

[55] K. Mowery, D. Bogenreif, S. Yilek, and H. Shacham. Fingerprinting information in JavaScript implementations. In *W2SP 2011, Oakland, CA, USA, May 26, 2011*, volume 2, pages 180–193, 2011. 7, 8

[56] K. Mowery and H. Shacham. Pixel perfect: Fingerprinting canvas in HTML5. In M. Fredrikson, editor, *W2SP 2012, San Francisco, CA, USA*. IEEE Computer Society, May 2012. 7, 11, 13, 29, 72

[57] M. Mulazzani, P. Reschl, M. Huber, M. Leithner, S. Schrittwieser, E. Weippl, and F. Wien. Fast and reliable browser identification with javascript engine fingerprinting. In *W2SP 2013, San Francisco, CA, USA, May 24, 2013*, volume 5, 2013. 1, 7

[58] A. Narayanan and D. Reisman. The Princeton web transparency and accountability project. In T. Cerquitelli, D. Quercia, and F. Pasquale, editors, *Transparent Data Mining for Big and Small Data*, volume 32 of *Studies in Big Data*, pages 45–67. Springer International Publishing, 2017. 14, 71, 79

[59] NetMarketShare. Browser market share. https://netmarketshare.com/browser-market-share.aspx. Accessed 25/02/2019. 77

[60] N. Nikiforakis, W. Joosen, and B. Livshits. Privaricator: Deceiving fingerprinters with little white lies. In A. Gangemi, S. Leonardi, and A. Panconesi, editors, *Proceedings of the 24th International Conference on World Wide Web, WWW 2015, Florence, Italy, May 18–22, 2015*, pages 820–830. ACM, 2015. 7, 69

[61] N. Nikiforakis, A. Kapravelos, W. Joosen, C. Kruegel, F. Piessens, and G. Vigna. Cookieless monster: Exploring the ecosystem of web-based device fingerprinting. In *2013 IEEE Symposium on Security and Privacy, SP 2013, Berkeley, CA, USA, May 19–22, 2013*, pages 541–555. IEEE Computer Society, 2013. 1, 13, 16, 17, 19, 20, 21, 30, 31, 66, 70

[62] N. Nikiforakis, A. Kapravelos, W. Joosen, C. Kruegel, F. Piessens, and G. Vigna. On the workings and current practices of web-based device fingerprinting. *IEEE Security & Privacy*, 12(3):28–36, June 2014. 68

[63] L. Olejnik, G. Acar, C. Castelluccia, and C. Díaz. The leaking battery — A privacy analysis of the HTML5 battery status API. In J. García-Alfaro, G. Navarro-Arribas, A. Aldini, F. Martinelli, and N. Suri, editors, *Data Privacy Management, and Security Assurance — 10th International Workshop, DPM 2015, and 4th International Workshop, QASA 2015, Vienna, Austria, September 21-22, 2015. Revised Selected Papers*, volume 9481 of *Lecture Notes in Computer Science*, pages 254–263. Springer, 2015. 7, 37, 73

[64] L. Olejnik, S. Englehardt, and A. Narayanan. Battery status not included: Assessing privacy in web standards. In *3rd International Workshop on Privacy Engineering (IWPE'17), San Jose, USA*, March 2017. 18, 19, 72

[65] M. Perry. Do not beg: Moving beyond DNT through privacy by design. W3C DNT, Tor Project, November 2012. https://www.w3.org/2012/dnt-ws/position-papers/21.pdf. 14, 70

[66] M. Perry, E. Clark, and S. Murdoch. The design and implementation of the tor browser. Draft, Tor Project, June 2013. https://www.torproject.org/projects/torbrowser/design. 2, 68, 71, 72, 73

[67] V. C. Perta, M. V. Barbera, G. Tyson, H. Haddadi, and A. Mei. A glance through the VPN looking glass: IPv6 leakage and DNS hijacking in commercial VPN clients. *PoPETs*, 2015(1):77–91, 2015. 47, 49

[68] G. Portokalidis, M. Polychronakis, A. D. Keromytis, and E. P. Markatos. Privacy-preserving social plugins. In T. Kohno, editor, *Proceedings of the 21th USENIX Security Symposium, Bellevue, WA, USA, August 8–10, 2012*, pages 631–646. USENIX Association, 2012. 8

[69] E. Rescorla. HTTP over TLS. RFC 2818, RFC Editor, May 2000. http://www.rfc-editor.org/rfc/rfc2818.txt. 23

[70] F. Roesner, T. Kohno, and D. Wetherall. Detecting and defending against third-party tracking on the web. In S. D. Gribble and D. Katabi, editors, *Proceedings of the 9th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2012, San Jose, CA, USA, April 25–27, 2012*, pages 155–168. USENIX Association, 2012. 8

[71] J. Rosenberg, R. Mahy, P. Matthews, and D. Wing. Session traversal utilities for NAT (STUN). RFC 5389, RFC Editor, October 2008. http://www.rfc-editor.org/rfc/rfc5389.txt. 12

[72] T. Sandholm, B. Magnusson, and B. A. Johnsson. An on-demand WebRTC and IoT device tunneling service for hospitals. In M. Younas, I. Awan, and A. Pescapè, editors, *2014 International Conference on Future Internet of Things and Cloud, FiCloud 2014, Barcelona, Spain, August 27–29, 2014*, pages 53–60. IEEE Computer Society, 2014. 12

[73] S. Schelter and J. Kunegis. On the ubiquity of web tracking: Insights from a billion-page web crawl. *J. Web Science*, 4(4):53–66, 2018. 27

[74] L. Simpkins, X. Yuan, J. Modi, J. Zhan, and L. Yang. A course module on web tracking and privacy. In M. E. Whitman and H. Zafar, editors, *Proceedings of the 2015 Information Security Curriculum Development Conference, InfoSecCD 2015, Kennesaw, GA, USA, October 10, 2015*, pages 10:1–10:7. ACM, 2015. 75

[75] D. Singer and R. Fielding. Tracking preference expression (DNT). W3C working group note, W3C, Jan. 2019. https://www.w3.org/TR/2019/NOTE-tracking-dnt-20190117. 70

[76] P. Snyder, L. Ansari, C. Taylor, and C. Kanich. Browser feature usage on the modern web. In *Proceedings of the 2016 ACM on Internet Measurement Conference,*

IMC 2016, Santa Monica, CA, USA, November 14–16, 2016, pages 97–110. ACM, 2016. 72

[77] O. Starov and N. Nikiforakis. XHOUND: quantifying the fingerprintability of browser extensions. In *2017 IEEE Symposium on Security and Privacy, SP 2017, San Jose, CA, USA, May 22–26, 2017*, pages 941–956. IEEE Computer Society, 2017. 70

[78] K. SZYMIELEWICZ and B. BUDINGTON. The gdpr and browser fingerprinting. https://www.eff.org/deeplinks/2018/06/gdpr-and-browser-fingerprinting-how-it-changes-game-sneakiest-web-trackers. Accessed 23/07/2019. 1

[79] N. Takei, T. Saito, K. Takasu, and T. Yamada. Web browser fingerprinting using only cascading style sheets. In L. Barolli, F. Xhafa, M. R. Ogiela, and L. Ogiela, editors, *10th International Conference on Broadband and Wireless Computing, Communication and Applications, BWCCA 2015, Krakow, Poland, November 4–6, 2015*, pages 57–63. IEEE Computer Society, 2015. 13

[80] C. F. Torres, H. L. Jonker, and S. Mauw. Fp-block: Usable web privacy by controlling browser fingerprinting. In G. Pernul, P. Y. A. Ryan, and E. R. Weippl, editors, *Computer Security — ESORICS 2015 — 20th European Symposium on Research in Computer Security, Vienna, Austria, September 21–25, 2015, Proceedings, Part II*, volume 9327 of *Lecture Notes in Computer Science*, pages 3–19. Springer, 2015. 65, 69

[81] J. Uberti and G. wei Shieh. WebRTC IP Address Handling Requirements. Internet-Draft draft-ietf-rtcweb-ip-handling-09, Internet Engineering Task Force, June 2018. https://datatracker.ietf.org/doc/html/draft-ietf-rtcweb-ip-handling-09. 49

[82] T. Unger, M. Mulazzani, D. Fruhwirt, M. Huber, S. Schrittwieser, and E. R. Weippl. SHPF: enhancing HTTP(S) session security with browser fingerprinting. In *2013 International Conference on Availability, Reliability and Security, ARES 2013, Regensburg, Germany, September 2–6, 2013*, pages 255–261. IEEE Computer Society, 2013. 8

[83] R. Upathilake, Y. Li, and A. Matrawy. A classification of web browser fingerprinting techniques. In M. Badra, A. Boukerche, and P. Urien, editors, *7th International Conference on New Technologies, Mobility and Security, NTMS 2015, Paris, France, July 27–29, 2015*, pages 1–5. IEEE, 2015. 7

[84] B. Ur, P. G. Leon, L. F. Cranor, R. Shay, and Y. Wang. Smart, useful, scary, creepy: perceptions of online behavioral advertising. In L. F. Cranor, editor, *Symposium On Usable Privacy and Security, SOUPS '12, Washington, DC, USA — July 11 – 13, 2012*, page 4. ACM, 2012. 13

[85] A. Vastel, P. Laperdrix, W. Rudametkin, and R. Rouvoy. FP-Scanner: The privacy implications of browser fingerprint inconsistencies. In W. Enck and A. P. Felt, editors, *27th USENIX Security Symposium, USENIX Security 2018, Baltimore, MD, USA, August 15–17, 2018*, pages 135–150. USENIX Association, August 2018. 70

[86] A. Vastel, P. Laperdrix, W. Rudametkin, and R. Rouvoy. FP-STALKER: tracking browser fingerprint evolutions. In *2018 IEEE Symposium on Security and Privacy, SP 2018, Proceedings, 21–23 May 2018, San Francisco, California, USA*, pages 728–741. IEEE, 2018. 1, 8

[87] WHATWG community. The canvas element. Living standard, Web Hypertext Application Technology Working Group, April 2019. https://html.spec.whatwg.org/multipage/canvas.html. 11

[88] T. Yen, Y. Xie, F. Yu, R. P. Yu, and M. Abadi. Host fingerprinting and tracking on the web: Privacy and security implications. In *19th Annual Network and Distributed System Security Symposium, NDSS 2012, San Diego, California, USA, February 5–8, 2012*. The Internet Society, 2012. 10

[89] B. Zhao and P. Liu. Private browsing mode not really that private: Dealing with privacy breach caused by browser extensions. In *45th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, DSN 2015, Rio de Janeiro, Brazil, June 22–25, 2015*, pages 184–195. IEEE Computer Society, 2015. 14

[90] S. Zimmeck, J. S. Li, H. Kim, S. M. Bellovin, and T. Jebara. A privacy analysis of cross-device tracking. In E. Kirda and T. Ristenpart, editors, *26th USENIX Security Symposium, USENIX Security 2017, Vancouver, BC, Canada, August 16–18, 2017*, pages 1391–1408. USENIX Association, 2017. 8

Appendices

This thesis concludes with a series of appendices containing supplementary information about the experiments described in Chapters 3, 4 and 5.

Appendix A gives details of the websites identified as performing fingerprinting in the crawl of 10,000 websites (described in Chapter 3).

Appendix B lists the Python script that was used to control the crawler discussed in Chapter 3.

Appendix C lists the 286 fingerprintable browser attributes (divided into six categories) that were discovered in the data collected in the experiment described in Chapter 3.

Appendix D lists the sources of the code used on the *fingerprintable* website as well as the HTML source code of the fingerprintability test page. This test page was used for the experiments described in Chapter 4.

Appendix E lists the specifications and versions of operating systems, browsers and devices used in the experiments described in Chapter 4.

Appendix F lists the sources of the code as well as the code used on `https://fingerprintable.org/webrtcleaks` for the experiments described in Chapter 5.

A Results of Crawling Experiment

A.1 Major Fingerprinters

Table 1 lists the fingerprinter domains that were present on at least 100 of the 10,000 websites tested as part of the experiment described in Chapter 3.

Table 1: Fingerprinters present on at least 100 websites

Number of Websites	Fingerprinter Domain
5,860	google-analytics.com
3,233	doubleclick.net
1,377	google.com
1,349	google.co.uk
1,117	googlesyndication.com
879	quantserve.com
528	rubiconproject.com
506	omtrdc.net
498	openx.net
473	bing.com
432	moatads.com
388	adsafeprotected.com
276	baidu.com
220	parsely.com
212	bluekai.com
197	pubmatic.com
172	2o7.net
172	criteo.com
150	cloudfront.net
138	googleadservices.com
123	cnzz.com
115	krxd.net
113	pinterest.com

A.2 Suspected Fingerprinter Domains

Listed below are all 1,914 domains that were recipients of detected browser fingerprinting information in the experiment described in Chapter 3.

192.61.11.116,218.145.26.75,5.45.67.97,165.194.95.13,106.3.131.15,115.182.217.24,01net.com,0catch.com,10010.com,10jqka.com.cn,11alive.com,12371.cn,126.net,110.93.143.144,15gifts.com,17u.cn,1gb.ru,1plus1.ua,1rx.io,21cn.com,247-inc.net,2gis.ru,2o7.net,33across.com,360.cn,360buyimg.com,39.net,3ders.net,3dprint.com,3gl.net,3m.com,40nuggets.com,4dsply.com,4players.de,50bang.org,51.com,51.la,51cto.com,51yes.com,6.cn,6rooms.com,99designs.com,9c9media.ca,9news.com,aa.com,aa.org,aa.org,aafp.org,aams

itecertifier.com,aao.org,aarp.org,abb.com,abc.net.au,abduzeedo.com,abebooks.com,abercrombie.co.uk,abercrombie.com,a-cast.jp,accenture.com,acecounter.com,acecounter.com,acehardware.com,acint.net,aclu.org,acquia.com,acronymfinder.com,acs.org,acs86.com,active.com,ad6media.fr,adage.com,adalyser.com,adapf.com,adasiaholdings.com,adbetnet.com,adbutler-alion.com,addroplet.com,adelement.com,adform.net,adfox.ru,adhigh.net,adikteev.com,adinall.com,adiode.com,adledge.com,adlmerge.com,adlooxtracking.com,admaster.com.cn,admaxim.com,admeira.ch,admixer.net,adnxs.com,adobe.com,adocean.pl,adrcntr.com,adriver.ru,adrta.com,adsafeprotected.com,ad-srv.net,adsrvr.org,ad-stir.com,adsupply.com,adswizz.com,adtarget.me,adtech.de,adtechjp.com,adtechus.com,adtelligence.de,adultswim.com,adventive.com,adventori.com,advertising.com,advinapps.com,adworx.at,adxvip.com,ae.com,aegpresents.com,aeroflot.ru,aetn.com,aetna.com,aetnamedicare.com,affinity.com,affirm.com,afsanalytics.com,aftonbladet.se,ageuk.org.uk,ahalogy.com,ahrq.gov,aig.com,airbnb.co.uk,aircanada.com,akamaihd.net,akro.io,akstat.io,alaskaair.com,albacross.com,al-consulting.com,alibaba.com,alimama.com,aller.fi,allrecipes.com,allstate.com,al-monitor.com,altitude-arena.com,altitudeplatform.com,alwaysdata.com,amazing-media.com,amazon.ca,amazon.cn,amazon.co.jp,amazon.co.uk,amazon.com,amazon.de,amazon.es,amazon.fr,amazon.in,amazon.it,amazon-adsystem.com,amazonaws.com,ambra.com,amcnetworks.com,amctv.com,americanairlines.co.uk,americanbar.org,americanexpress.com,amplitude.com,analog.com,analyticsbridge.io,analyze.ly,ancestry.com,and.co.uk,angsrvr.com,animoto.com,anrdoezrs.net,antenna.is,anthropologie.com,ants.vn,anyclip-media.com,aol.com,apa.org,app.com,appboy.com,apple.com,appspot.com,apptentive.com,aptrackr.com,aqtracker.com,arbeitsagentur.de,architecturaldigest.com,archive-it.org,arcpublishing.com,areyouahuman.com,argos.co.uk,art.com,aruba.it,asics.com,asme.org,asos.com,astd.org,asus.com,atatus.com,atdmt.com,atemda.com,ati-host.net,atlanticinsights.com,atp.io,att.com,att.net,attributionapp.com,au.com,audi.com,audible.co.uk,audienceinsights.net,augsburger-allgemeine.de,auspost.com.au,autobild.de,autodesk.com,autohome.com.cn,automobilemag.com,autonews.com,autoscout24.com,av.st,avg.com,aviationweek.com,axs.com,azalead.com,azcentral.com,b2b168.com,babator.com,babycenter.com,babytree.com,backcountry.com,baden-wuerttemberg.de,badoo.com,bahn.de,baicai.cn,baidu.com,baifendian.com,baike.com,baltimoresun.com,bam-x.com,banggood.com,bankofamerica.com,bankrate.com,barclays.co.uk,barilliance.net,basf.com,basspro.com,bayern.de,bazaarvoice.com,bbc.co.uk,bbelements.com,beatsbydre

.com,bedbathandbeyond.com,beemray.com,belfasttelegraph.co.uk,bell.ca,be
mobile.ua,benetton.com,beopinion.com,bergdorfgoodman.com,berlinonline.d
e,bestadbid.com,bestbuy.ca,bestbuy.com,bet.com,betweendigital.com,bfmio
.com,bfmtv.com,bhphotovideo.com,bidfluence.com,bidswitch.net,bihk.de,bi
ld.de,billingsgazette.com,bing.com,bing-int.com,bitauto.com,bitdefender
.com,blog.de,blog.hu,blogads.com,blogg.se,blogher.com,blogmura.com,bloo
mingdales.com,blueconic.net,bluehost.com,bluekai.com,bmbfcluster.de,bmm
etrix.com,bmo.com,bna.com,bodybuilding.com,bol.com,bonappetit.com,bookd
epository.com,booking.com,bose.com,boston.com,bostonglobemedia.com,boun
ceexchange.com,bouncex.net,boutell.com,bpb.de,brandcrumb.com,brides.com
,brightcove.com,brightfunnel.com,brighthub.com,brightinfo.com,britishai
rways.com,brocade.com,brookstone.com,brownpapertickets.com,bt.com,bttta
g.com,bugsnag.com,burlingtonfreepress.com,buzzfeed.com,bzgint.com,c3tag
.com,ca.com,cabelas.com,caesars.com,cafe24.com,caixa.gov.br,calgaryhera
ld.com,callrail.com,canadapost.ca,canal-plus.com,canoe.com,canopylabs.c
om,canopylabs.com,capcom.co.jp,capitalone.com,capturehighered.net,caree
ronestop.org,carnival.com,carrierzone.com,castle.io,cbc.ca,cbn.com,cbsi.
com,cbsistatic.com,cc.com,ccb.com,c-ctrip.com,cdlib.org,cdnwebcloud.com,
cdvcloud.com,cdw.com,ceair.com,cedexis.com,centurylink.com,cerebroad.co
m,change.org,changeagain.me,channel4.com,charter.com,chase.com,chatpath.
com,checkm8.com,chegg.com,cheshi.com,chicagobusiness.com,chicagotribune.
com,chinaacc.com,chinanews.com,chip.de,chitika.net,chowhound.com,christ
ies.com,chronicle.com,cigna.com,cihar.com,cincinnati.com,cisco.com,citi
.com,citicards.com,citizen-times.com,citrix.com,clarionledger.com,click
ability.com,clickiocdn.com,clicktale.net,clicrbs.com.br,cliqz.com,clnmd
e.com,cloudfront.net,clrstm.com,cmt.com,cmu.edu,cnet.com,cnfol.com,cngo
ld.org,cnil.fr,cnixon.com,cnn.com,cnrs.fr,cntvwb.cn,cnwest.com,cnzz.com
,co.kr,codeproject.com,coe.int,cohesionapps.com,collegeboard.org,com.cn
,come.to,comm100.com,commercesciences.com,commercialappeal.com,comodo.c
om,compuserve.com,computerbild.de,concurra.com,condenast.com,condenastd
igital.com,congress.gov,connexity.net,constant.co,constantcontact.com,c
onstitution.org,consultant.ru,consumerreports.org,contently.com,content
sfeed.com,contentsquare.net,contextweb.com,converse.com,conversionlogic.
net,convertlanguage.com,convertro.com,coremetrics.com,corriere.it,corus.
ca,courant.com,courier-journal.com,coursera.org,cox.com,cqdailynews.com,
crainsnewyork.com,crateandbarrel.com,craveonline.com,creativemarket.com
,creditcards.com,credit-suisse.com,criteo.com,crobox.com,crowdskout.com

,crunchbase.com,crutchfield.com,crwdcntrl.net,csair.com,csbew.com,csmon
itor.com,csu.edu.cn,ctags.cn,ctnsnet.com,ctrip.com,ctv.ca,customer.io,c
ustora.com,cvent.com,cvshealth.com,cxense.com,daad.de,daemon-tools.cc,d
ailymotion.com,dailypress.com,dailyprogress.com,dangdang.com,danzz.ch,d
aringfireball.net,datacenterknowledge.com,dc-storm.com,debenhams.com,de
ep.bi,deepsight.io,defense.gov,delawareonline.com,delfi.ee,delfi.lv,del
iverimp.com,dell.com,deloitte.com,delta.com,deluxe.com,demdex.net,democ
ratandchronicle.com,deployads.com,desmoinesregister.com,detroitnews.com
,deutsche-bank.de,df-srv.de,dhm.de,dialogtech.com,dianping.com,dice.com,
dickssportinggoods.com,diesel.com,digicert.com,digikey.com,digitaltrend
s.com,directv.com,discover.com,discoverhongkong.com,diynetwork.com,dm.g
g,dmm.com,dmp.org.cn,dmtracker.com,dmtry.com,dmv.org,dnb.com,do-analyti
cs.net,doctoroz.com,documentfoundation.org,dotomi.com,doubleclick.net,d
oublepimp.com,doublepimpssl.com,doubleverify.com,dowjoneson.com,drift.c
om,dtic.mil,duowan.com,dxc.technology,dxy.cn,dynamicyield.com,dynatrace.
com,dynatrace-managed.com,dynatracesaas.com,dyntrk.com,easeus.com,eastm
oney.com,easy-ads.com,easyjet.com,ebay.com,ebrun.com,ebu.io,ecn.cl,econ
omist.com,ecustomeropinions.com,edgesuite.net,edigitalsurvey.com,edmont
onjournal.com,edmunds.com,edu.co,edu.sg,edweek.org,ee.co.uk,effectiveme
asure.net,egovsum.com,ehawaii.gov,eiu.com,element14.com,elignum.net,elm
undo.es,el-mundo.net,elong.com,elongstatic.com,elsevier.com,eltiempo.co,
eluniversal.com,eluniversal.com.mx,emarbox.com,emarketer.com,emc.com,em
irates.com,engadget.com,enorth.com.cn,enterprise.com,eol.cn,eonline.com,
epicurious.com,epo.org,eproof.com,equifax.com,eqxiu.com,esearchvision.c
om,espn.com,estara.com,estat.com,ethz.ch,etihad.com,etracker.de,ets.org
,etsy.com,eum-appdynamics.com,eventim.de,everesttech.net,everydayhealth.
com,exoclick.com,exosrv.com,expansion.com,expedia.co.uk,expedia.com,exp
onea.com,express.co.uk,extreme-dm.com,extremetech.com,facebook.com,fair
mont.com,faithlifeads.com,famfamfam.com,familysearch.org,fandango.com,f
arfetch.com,farfetch.net,farsnews.com,fastapi.net,fastly.net,fau.de,fbc
dn.net,fc2web.com,fcacert.com,feathr.co,federalreserve.org,fedex.com,fe
edjit.com,feedsportal.com,fender.com,fengniao.com,fes.de,ffx.io,fidelit
y.com,fifa.com,filemaker.com,financialpost.com,findify.io,findlaw.com,f
inland.fi,finnair.com,firstcoastnews.com,firstimpression.io,fivetran.co
m,flashtalking.com,floridatoday.com,flow.io,flysas.com,fnac.com,focus.d
e,foodnetwork.com,forbes.com,ford.com,foreignpolicy.com,foresee.com,for
milla.com,forward.com,fotolia.net,foxnews.com,foxsports.com,fqtag.com,f

rancetelecom.com,freecreditreport.com,freelancer.com,freemake.com,freep
.com,freepeople.com,freiheit.org,fs-bdash.com,fuel451.com,fullstory.com,
funnelenvy.com,funshion.com,futurelearn.com,fwmrm.net,g4tv.com,gamestop.
com,gap.com,garmin.com,gatesfoundation.org,gatesnotes.com,gator.io,gaug.
es,gazeta.pl,gazzetta.it,gcimetrics.com,geek.com,gehealthcare.com,gemiu
s.pl,genieesspv.jp,geniuskitchen.com,getambassador.com,getclicky.com,ge
tresponse.com,getsentry.com,gettyimages.com,gh-base.com,gibson.com,giga
zine.net,gilt.com,giosg.com,github.com,githubapp.com,globalsecurity.org
,glotgrx.com,gmossp-sp.jp,gnome.org,gnpge.com,gnpiwik.com,go.com,goarmy.
com,godaddy.com,golem.de,golfdigest.com,gome.com.cn,gomeplus.com,google.
co.uk,google.com,googleadservices.com,google-analytics.com,googleapis.c
om,googlesyndication.com,gosquared.com,gostats.com,gov.au,gov.uk,graing
er.com,grammarly.com,grammarly.io,granify.com,gravity.com,gridsumdissec
tor.com,groupon.com,grundfos.com,gtags.net,guidestar.org,guitarcenter.c
om,gumgum.com,gwallet.com,gwdg.de,haaretz.co.il,hadvid.com,haiyunpush.c
om,hallmark.com,hankooki.com,hao123.com,harborfreight.com,harley-davids
on.com,harrods.com,hbo.com,hc360.com,headspace.com,healthcentral.com,he
althgrades.com,heathrow.com,heias.com,heise.de,heraldcorp.com,herokuapp
.com,hexagon-analytics.com,hgtv.com,hiido.com,hilton.com,hit.ua,hitslin
k.com,hitsprocessor.com,hjapi.com,hktdc.com,hm.com,holland.com,homeadvi
sor.com,homedepot.com,honda.com,hongxiu.com,hoodin.com,hoovers.com,hory
zon-media.com,hostelbookers.com,hostelworld.com,hostinger.io,hostmonste
r.com,hotelgroup.com,hotels.com,hotelsapi.io,hotjar.com,hotlog.ru,hotsc
ripts.com,hp.com,hpe.com,hub.com.pl,hubpd.com,hubspot.com,hudong.com,hu
ffingtonpost.com,hugoboss.com,hujiang.com,hulu.com,hupu.com,hurpass.com
,hurriyet.com.tr,huxiu.com,hyatt.com,i.ua,iberia.com,ibtimes.com,icloud.
com,icm.edu.pl,idgesg.net,ifeng.com,iheart.com,ihg.com,ihs.com,ikea.com,
ilsole24ore.com,imagineinspired.com,imf.org,imgur.com,immobilienscout24
.de,imninjas.com,imonomy.com,impact-ad.jp,imrworldwide.com,in.th,indepe
ndent.co.uk,independent.ie,index.ru,indiegogo.com,indigo.ca,industrywee
k.com,indystar.com,infinity-tracking.net,infolinks.com,informz.net,infu
sionsoft.com,inist.fr,inq.com,inria.fr,inside-graph.com,inskinad.com,in
stagram.com,instantssl.com,intel.com,intentmedia.net,intergi.com,interg
ient.com,interia.pl,intuit.com,investors.com,invoca.net,invodo.com,ioam
.de,iocnt.net,iogous.com,ipinyou.com,iprom.net,ireland.com,irib.ir,iris
hcentral.com,irishtimes.com,irna.ir,ironad.de,istockphoto.com,itamaraty
.gov.br,itu.int,itzbund.de,iubenda.com,ivcbrasil.org.br,ixbt.com,ixquic

k.com,janrain.xyz,jaywing.com,jbl.com,jc001.cn,jcrew.com,jd.com,jetblue
.com,jetstar.com,jia.com,jiameng.com,jiemian.com,joann.com,johnlewis.co
m,joins.com,journalnow.com,journalstar.com,journity.com,jsonline.com,js
tor.org,juicer.cc,juicer.io,jumei.com,juntadeandalucia.es,justgiving.co
m,justhost.com,justpremium.com,justuno.com,jyu.fi,kaercher.com,kaipuyun
.cn,kaiserpermanente.org,kaixin001.com.cn,kandle.org,kankanews.com,kare
11.com,kas.de,kasperskycontenthub.com,kbb.com,kck.st,kde.org,keybase.io,
keytiles.com,keywee.co,kfw.de,kgw.com,kharkov.ua,khou.com,kicker.de,kij
iji.ca,king5.com,kiosked.com,kissmetrics.com,kmart.com,km-sea.net,knoxn
ews.com,kochava.com,kohls.com,komoona.com,koolearn.com,kpmg.com,kpn.com
,kroger.com,krxd.net,ksdk.com,kugou.com,kurier.at,kvue.com,labocleo.org
,lacoste.com,ladepeche.fr,lancasteronline.com,landsend.com,lanl.gov,lap
topmag.com,latrobe.edu.au,law.com,lawyers.com,lds.org,leadboxer.com,lea
dforensics.com,leadformix.com,leanplum.com,lego.com,leju.com,lenovo.com
,letv.com,lexpress.fr,lexus.com,lg.com,liadm.com,liberation.fr,libertym
utual.com,lifehack.org,lightstep.com,linkedin.com,linkfire.com,liqwid.n
et,list.ru,litix.io,livehelpnow.net,livescience.com,livingsocial.com,lk
qd.net,llbean.com,llnl.gov,llnw.net,loc.gov,lofter.com,logentries.com,l
ogger.co.kr,loggly.com,loginside.co.kr,lohud.com,lotour.com,louisvuitto
n.com,lowes.com,lrb.co.uk,ltn.com.tw,lufthansa.com,lululemon.com,luxupa
dva.com,luxupcdna.com,luxupcdnc.com,lww.com,lytics.io,macobserver.com,m
acys.com,madison.com,mafengwo.cn,magazinereadermall.com,mail.ru,majesti
c.com,makepolo.com,makeuseof.com,malaysiaairlines.com,mapmylead.com,map
smarker.com,mapyourshow.com,marcjacobs.com,marketingautomation.services,
marketingpower.com,marksandspencer.com,marriott.com,mars.com,marvel.com,
marykay.com,mashable.com,mastercard.com,matheranalytics.com,mathtag.com
,mathworks.com,matraxis.net,maxthon.com,maxymiser.net,mbmedien.de,mcall.
com,mdanderson.org,mdctrail.com,med66.com,media.net,mediapostcommunicat
ion.net,mediatoday.ru,mediav.com,mediawayss.com,medicaldaily.com,medium.
com,medpagetoday.com,medtronic.com,meilele.com,meituan.com,meizu.com,me
lia.com,member-hsbc-group.com,mercadolibre.com,mercer.com,mercure.com,m
erkur.de,metacritic.com,metrics-shell.com,metro.co.uk,metrolyrics.com,m
etronews.ca,mezzobit.com,mgtv.com,mheducation.com,mi.com,miaozhen.com,m
ichaelkors.co.uk,michaels.com,microsoft.com,milb.com,military.com,mille
nniumhotels.com,mindbox.ru,mirror.co.uk,missoulian.com,ml.com,ml314.com,
mlb.com,mmapiws.com,mmstat.com,moatads.com,mobicast.io,mobile.de,modclo
th.com,modernhealthcare.com,mogujie.com,momtastic.com,monarchads.com,mo

ndediplo.com,monde-diplomatique.fr,monetate.net,monsido.com,monster.com,
montrealgazette.com,mop.com,morganstanley.com,morningstar.com,motigo.co
m,motortrend.com,mouseflow.com,mouser.com,moveaws.com,moviefone.com,moz
.com,mpg.de,mpnrs.com,mql5.com,mrpfd.com,mrporter.com,msecnd.net,msgapp.
com,msn.com,msu.edu,mtime.cn,mtv.com,murdoch.edu.au,musee-orsay.fr,mway
ss.com,mycounter.ua,mydatabankapp.com,myfinance.com,myhsw.cn,mytopf.com,
mywebstats.eu,nakanohito.jp,nanovisor.io,naplesnews.com,nasa.gov,nascar
.com,nasdaq.com,natify.io,nationalpost.com,nationbuilder.com,nature.org
,naver.com,naver.jp,nba.com,nbcuas.com,nbcuni.com,ncaa.com,neimanmarcus
.com,nejm.org,neogaf.com,nero.com,nervoussummer.com,net-a-porter.com,ne
tflix.com,netmng.com,newbalance.com,newcouponuk.info,newegg.com,news.cn
,news.co.uk,news10.net,newsarama.com,newscgp.com,newsinc.com,news-leade
r.com,news-press.com,newyorker.com,next.co.uk,nextuser.com,nexusmods.co
m,nfl.com,nfpa.org,ngacm.com,nhs.uk,ni.com,nielsen.com,nih.gov,nike.com,
nikkeibp.co.jp,nine.com.au,nintendo.com,nissanusa.com,nj.us,nmgnews.com.
cn,nordstrom.com,nordstromdata.com,northeastern.edu,northernandshell.co
.uk,northjersey.com,norton.com,nr-data.net,ns8ds.com,nt.vc,nvidia.com,n
witimes.com,nxp.com,nycgo.com,nycgovparks.org,nymag.com,nytimes.com,nzz
.ch,o2.co.uk,ocdn.eu,oed.com,oewabox.at,offcn.com,officedepot.com,ogsta
tic.com,omaha.com,omnigroup.com,omnitagjs.com,omtrdc.net,on.cc,onecount
.net,onenote.com,onet.pl,onet.tv,online.net,onscroll.com,onthe.io,opecl
oud.com,openlibrary.org,openstat.net,openstreetmap.org,opensuse.org,ope
ntable.com,opentext.com,opentracker.net,openx.net,openxmarket.jp,opinio
nlab.com,oprah.com,optimix.asia,optimizely.com,optkit.com,ora.tv,orange
.fr,orbitz.com,orcinus.ai,oreilly.com,ornl.gov,os-data.com,osnews.com,o
sxdaily.com,otracking.com,ottawacitizen.com,otto.de,outbrain.com,overcl
ock.net,overstock.com,owncloud.org,owox.com,oxfam.org.uk,ozon.ru,paddle.
com,panasonic.com,pandora.net,panerabread.com,parker.com,parsely.com,pa
stebin.com,patagonia.com,paypal.com,pcauto.com.cn,pcbaby.com.cn,pchouse.
com.cn,pclady.com.cn,pconline.com.cn,pennwell.com,people.com,pepsico.co
m,perfdrive.com,permutive.com,persgroep.net,petametrics.com,petco.com,p
etsmart.com,petsmartdmz.com,pga.com,philanthropy.com,picreel.com,piloto
nline.com,pingan.com,pingdom.net,pinterest.ca,pinterest.co.uk,pinterest
.com,pitchfork.com,piwigo.us,piwik.org,pixanalytics.com,pixfuture.net,p
lala.or.jp,playstation.com,plista.com,plug.it,po.st,poco.cn,pointillist.
com,politico.com,politico.eu,popads.net,popsci.com,popsugar.com,positiv
essl.com,postandcourier.com,potterybarn.com,powster.com,poznan.pl,prada

.com,preamp.io,presidencia.gov.br,pressofatlanticcity.com,pressreader.com,priceline.com,princess.com,pro6e.com,probtn.com,proper.io,protecmedia.com,provenpixel.com,proxad.net,pubmatic.com,pubwise.io,puma.com,pushwoosh.com,q1connect.com,qantas.com,qatarairways.com,qchannel03.cn,qctimes.com,qidian.com,qq.com,qsstats.com,qtmojo.com,qualtrics.com,quantserve.com,queit.in,qunar.com,qut.edu.au,qvc.com,qyer.com,rakuten.co.jp,rakuten.com,ralphlauren.com,rambler.ru,randomhouse.com,ranker.com,ray-ban.com,raygun.io,raytheon.com,rbc.ru,rcsmetrics.it,readnovel.com,realclearpolitics.com,realestate.com.au,realtor.com,realvu.net,rebsrv.tk,redcross.org,redcrossblood.org,reddit.com,redfin.com,rediff.com,redplanetgroup.com.au,reebok.co.uk,reedbusiness.net,reevoo.com,register.com,reitingas.lt,renewableenergyworld.com,report-uri.com,report-uri.io,repubblica.it,research.gov,researchandmarkets.com,research-int.se,responsetap.com,responsiveads.com,retailmenot.com,retentionscience.com,rfihub.com,rgj.com,richaudience.com,richmetrics.com,richmond.com,ricoh.co.jp,rijksoverheid.nl,ringier.ch,ripe.net,rlp.de,rnet.plus,roanoke.com,rogers.com,rogersmedia.com,rolex.com,rollbar.com,romsenergy.com,rosettastone.com,royalcaribbean.com,royalmail.com,rphelios.net,rp-online.de,rtb-media.me,rtm.com,rub.de,rubiconproject.com,ryanair.com,s6w.de,sa-as.com,sacbee.com,sacbeelabs.com,safer-networking.ie,saksfifthavenue.com,salecycle.com,salesforce.com,salesgenius.com,samsclub.com,samsung.com,sandiegouniontribune.com,sanoma.fi,sas.com,savethechildren.org,saveur.com,sberbank.ru,schibsted.io,scholarlyiq.com,schwab.com,sciencedaily.com,sciencemag.org,scorecardresearch.com,scotiabank.com,scroll.com,searchlinks.com,sears.com,seattlepi.com,securedvisit.com,secureserver.net,securityfocus.com,seek.com,seek.com.au,segment.io,sekindo.com,selectmedia.asia,self.com,selfcampaign.com,sentry.io,seobook.com,servedbyadbutler.com,servedbyopenx.com,serving-sys.com,sessioncam.com,sfr.fr,sgs.com,shapeways.com,sharecare.com,sharefile.com,sharethrough.com,sherdog.com,sherwin-williams.com,shinystat.com,shoprunner.com,shopstyle.co.uk,shutterfly.com,siftscience.com,sigmaaldrich.com,signalvnoise.com,similarweb.com,simplymeasured.com,sina.cn,sina.com.cn,singaporeair.com,siriusxm.com,sitecues.com,siteimprove.com,sketchfab.com,skimresources.com,sky.com,sky.it,skyscanner.net,sl.pt,slacker.com,slashdotmedia.com,slate.com,smartadserver.com,smartertravel.com,smartlook.com,smashingmagazine.com,s-microsoft.com,smithsonian.museum,smugmug.com,snapfish.com,snoobi.com,socialhoney.co,softonic-analytics.net,sogou.com,sohu.com,so-net.ne.jp,sonos.com,sony.jp,sonypictures.com

,sophus3.com,sothebys.com,southwest.com,space.com,spartanrtb.com,spbu.r
u,specless.io,spectate.com,spectrum.com,spiceworks.com,spie.org,spike.c
om,splunkcloud.com,sportingnews.com,sports.ru,spotxchange.com,spreadshi
rt.com,spreadshirt.net,springserve.com,spring-tns.net,sprint.com,sputni
k.ru,spylog.com,squarespace.com,squareup.com,sspinc.io,ssrn.com,st.com,
standardandpoors.com,starpulse.com,startribune.com,starwoodhotels.com,s
tatcounter.com,statefarm.com,statesmanjournal.com,staticstuff.net,stats
event.com,stellarium.org,stjude.org,stltoday.com,storetail.io,stormcont
ainertag.com,stormiq.com,strato.de,streamrail.com,streamrail.net,streem
.com.au,streetinsider.com,stridespark.com,strikingly.com,stripe.com,stu
bhub.com,study.com,stuff.co.nz,stuttgarter-zeitung.de,stylemepretty.com
,styria-digital.com,sueddeutsche.de,summerhamster.com,sumo.com,suning.c
om,sun-sentinel.com,supportindeed.com,survicate.com,suunto.com,svd.se,s
vtrd.com,swarovski.com,sway.com,swiss.com,switchadhub.com,sydney.edu.au,
sylon.net,symantec.com,synopsys.com,t3n.sc,taboola.com,tagesanzeiger.ch
,tagtic.cn,tailsweep.com,tallahassee.com,tampabay.com,taobao.com,target
.com,tcm.com,td.com,tda.io,te.com,tealiumiq.com,teamcoco.com,techcrunch.
com,techinasia.com,techweb.com.cn,teenvogue.com,telegraph.co.uk,telenet
.be,telestream.net,telia.se,telllwrite.com,telstra.com.au,telus.com,ten
nessean.com,tenpay.com,terra.com.br,tes.com,tesco.com,test.de,testin.cn,
theadex.com,theadvocate.com,thebigwillow.work,thebrighttag.com,thefreed
ictionary.com,thefreelibrary.com,theglobeandmail.ca,theglobeandmail.com,
thehindu.co.in,theinformation.com,theintercept.com,theoreminc.net,thepe
titionsite.com,theprovince.com,thermofisher.com,thestar.com,thinkgeek.c
om,thinkvine.com,thomson.co.uk,thomsonreuters.com,tianpeng.com,ticketma
ster.com,tiffany.com,tim.it,time.com,timeout.com,timeshighereducation.c
om,timewarnercable.com,tingyun.com,tinyurl.com,t-mobile.com,tmz.com,tns
-cs.net,tns-gallup.dk,to8to.com,tom.com,tomonline-inc.com,tomsguide.com,
tomshardware.com,top.ge,topshop.com,topsrvimp.com,toptenreviews.com,tot
albeauty.com,tourmake.it,townnews365.com,toysrus.com,trackad.cz,tracked
web.net,trackjs.com,tradelab.fr,trafficjunky.net,travelchannel.com,trav
elocity.com,trb.com,treasury.gov,trello.com,tremorhub.com,trendmicro.co
m,trib.com,tribalfusion.com,triblive.com,trs.cn,trugaze.io,trulia.com,t
rustedreviews.com,trustpilot.com,truvidplayer.com,tsite.jp,tsn.ua,tso.c
o.uk,ttlbd.net,tu-berlin.de,tu-bs.de,tucson.com,tu-dresden.de,tugraz.at
,tui.co.uk,tulsaworld.com,tum.de,tuniu.cn,tuniu.com,turkishairlines.com
,turner.com,tuwien.ac.at,tv.com,tv2.dk,tvguide.com,tvn.pl,twcc.com,twea

kers.nl,twitch.tv,twitter.com,typesquare.com,ua.com,ub.edu,uber.com,ubi
.com,uc.cn,ucdenver.edu,uclouvain.be,udemy.com,udimg.com,ufc.com,ufrj.b
r,uibk.ac.at,uio.no,uiowa.edu,ui-portal.com,ui-portal.de,uk2group.com,u
nblog.fr,underarmour.co.uk,uni-bielefeld.de,unibo.it,uni-bremen.de,unic
a.com,unicc.org,unice.fr,uni-halle.de,uni-hannover.de,uni-heidelberg.de
,uni-jena.de,uni-koeln.de,uni-konstanz.de,uni-leipzig.de,uni-marburg.de,
uni-muenster.de,uni-oldenburg.de,uni-paderborn.de,uniqlo.com,uni-saarla
nd.de,uni-stuttgart.de,united.com,united-domains.de,uni-tuebingen.de,un
i-wuerzburg.de,unl.edu,unrulymedia.com,unsplash.com,uplift-platform.com,
upscore.io,uptolike.com,upwork.com,uralweb.ru,urbanoutfitters.com,urchi
n.com,usatoday.com,usbank.com,usehero.com,userreplay.net,userreport.com
,usopen.org,uspech.sk,ustream.tv,utdstc.com,utu.fi,uvic.ca,uzh.ch,uzone
.id,vancouversun.com,vanityfair.com,vast.com,vdopia.com,veinteractive.c
om,velaro.com,vemba.io,venatusmedia.com,ventunotech.com,verifystore.com
,verizon.com,verizonwireless.com,versace.com,vevo.com,vh1.com,viadeo.co
m,viafoura.co,viator.com,victoriassecret.com,videolan.org,vidible.tv,vi
dyard.com,viostream.com,viralize.tv,virginmoneygiving.com,virgul.com,vi
siblemeasures.com,visualstudio.com,visualwebsiteoptimizer.com,vivocha.c
om,vizual.ai,vmall.com,vmware.com,vodafone.co.uk,voice-of-customers.com,
volkswagen.com,voltairenet.org,volvogroup.com,voxmedia.com,voyages-sncf.
com,vsco.co,vtracy.de,vueling.com,vulture.com,vw.com,w3t.cn,waitrose.co
m,walgreens.com,walkme.com,walmart.com,wanmei.com,washingtonpost.com,wb
ir.com,weather.com,webdissector.com,webeyez.com,webglstats.com,webhosti
ngtalk.com,webmd.com,weborama.fr,webspectator.com,webterren.com,webtrad
ecenter.com,webtrekk.net,webtrends.com,webtrendslive.com,weebly.com,wei
bo.com,weiyun.com,weizmann.ac.il,wemfbox.ch,westelm.com,westernunion.co
m,wetransfer.net,wfaa.com,whatculture.com,wheretowatch.com,whitepages.c
om,widengle.com,wikia-services.com,wikihow.com,wiley.com,williamhill.co
m,williams-sonoma.com,wimbledon.com,windows.net,windowsitpro.com,windso
rcircle.com,wired.com,wistia.com,wizzair.com,wkyc.com,wmagazine.com,wmg
.com,wnba.com,woopra.com,worldcat.org,wowhead.com,wp.com,wp.pl,wral.com,
wrating.com,wraug5vv72b28fch.pro,wsod.com,wsws.org,wt-eu02.net,wtsp.com,
wunderground.com,wusa9.com,wvgazettemail.com,www.com,wyborcza.pl,wyndha
mhotels.com,xara.com,xcar.com.cn,xdf.cn,xerox.com,xing.com,xinmin.cn,xi
ti.com,xpanama.net,xplosion.de,xs4all.net,yabidos.com,yadro.ru,yadvashe
m.org,yahoo.co.jp,yahoo.com,yandex.com,yandex.ru,yandex.ua,yccdn.com,ye
ll.com,yellowpages.com,yelp.ca,yelp.co.uk,yelp.com,yesky.com,yieldify.c

om,yieldmo.com,yieldoptimisers.net,yldbt.com,ymetrica.com,yoka.com,yotpo.com,yottaa.com,yottaa.net,youboy.com,youdao.com,youravon.com,yourdomain.com,youtube.com,yum.de,yumenetworks.com,zarget.com,zbj.com,zcominc.com,zcool.com.cn,zeit.de,zero.kz,zhaopin.com,zhcw.com,zhubajie.la,zocdoc.com,zol.com.cn,zoosnet.net,zt51la.net,zz123.com

B Crawler Script

Below is the Python script used to control the crawling process discussed in Chapter 3.

```
import argparse
from urllib.parse import urlparse
import logging
import os
import time
from selenium import webdriver
from selenium.webdriver.chrome.options import Options
from selenium.common.exceptions import TimeoutException
import coloredlogs
logger = logging.getLogger()
coloredlogs.install(level='DEBUG')
def crawl_websites(start, end):
chrome_options = Options()
prefs = {"download.default_directory": "/Crawling/crawldata/"}
chrome_options.add_experimental_option("prefs", prefs)
chrome_options.add_extension('/Users/nasser/Desktop/Crawling/
    ↪ FingerprintPost.crx')
driver = webdriver.Chrome(chrome_options=chrome_options)
driver.set_page_load_timeout(60)
timeout_list = []
for i, line in enumerate(open("top_10000_websites.txt", "r")):
url = line.strip()
if i < start:
continue
if i > end:
logger.debug("Stopped at {0} website".format(i))
os.system(
"say 'For hes a jolly good fellow for hes a jolly good fellow! For hes
    ↪ a jolly good fellow which nobody can deny!'")
getConsent = input("Done!")
if getConsent.lower() == "go":
stop = True
elif getConsent.lower() == "no":
stop = False
```

```
if stop:
driver.quit()
try:
driver.get(url)
except TimeoutException as e:
logger.debug(e)
timeout_list.append(url)
with open("timeout_websites.txt", "a") as f:
for website in timeout_list:
f.write(website + "\n")
logger.debug("Stopped at {0} website".format(i))
os.system(
"say 'Its me the crawler! Please restart at {0}'".format(i))
os.system("say 'you know what to do'")
getConsent = input("Do you want to stop the crawler, Yes or no? ")
if getConsent.lower() == "yes":
stop = True
elif getConsent.lower() == "no":
stop = False
if stop:
driver.quit()
finally:
time.sleep(3)
def crawl(url):
chrome_options = Options()
chrome_options.add_extension('/Users/nasser/Desktop/Crawling/
    ↪ FingerprintPost.crx')
driver = webdriver.Chrome(chrome_options=chrome_options)
driver.set_page_load_timeout(5)
try:
driver.get(url)
except Exception as e:
logger.debug(e)
driver.set_script_timeout(3)
driver.execute_script("window.stop();")
try:
driver.set_page_load_timeout(30)
```

```
driver.post("http://google.com")
except Exception as e:
logger.debug(e)
else:
driver.close()
logger.debug("Process completes!")
def main():
parser = argparse.ArgumentParser(usage="python3 auto.py -s <number of
    ↪ websites to start> -e <number of websites to end>",
description='Extract main contents from a video file!!')
parser.add_argument("-s", "--start", type=int, dest="start",
help="the n websites to start!")
parser.add_argument("-e", "--end", type=int, dest="end",
help="the n websites to end!")
args = parser.parse_args()
s = args.start
e = args.end
crawl_websites(s, e)
if __name__ == '__main__':
main()
```

C Attributes Collected by Fingerprinters

This appendix lists all the fingerprintable browser attributes (divided into six categories) we were able to detect in the experiment described in Chapter 3.

C.1 WebGL

The following attributes are related to the WebGL API.

aliasedlinewidthrange,aliasedpointsizerange,alphabits,angleinstanced arrays,antialiasing,bluebits,depthbits,experimental-webgl,extblendminma x,extdisjointtimerquery,extfragdepth,extshadertexturelod,extsrgb,exttex turefilteranisotropic,fragmentshaderhighfloatprecision,fragmentshaderhi ghfloatprecisionrangemax,fragmentshaderhighfloatprecisionrangemin,fragm entshaderhighintprecision,fragmentshaderhighintprecisionrangemax,fragme ntshaderhighintprecisionrangemin,fragmentshaderlowfloatprecision,fragme ntshaderlowfloatprecisionrangemax,fragmentshaderlowfloatprecisionrangem in,fragmentshaderlowintprecision,fragmentshaderlowintprecisionrangemax, fragmentshaderlowintprecisionrangemin,fragmentshadermediumfloatprecisio n,fragmentshadermediumfloatprecisionrangemax,fragmentshadermediumfloatp recisionrangemin,fragmentshadermediumintprecision,fragmentshadermediumi ntprecisionrangemax,fragmentshadermediumintprecisionrangemin,greenbits, max3dtexturesize,maxanisotropy,maxarraytexturelayers,maxcolorattachment s,maxcombinedfragmentuniformcomponents,maxcombinedtextureimageunits,max combinedvertexuniformcomponents,maxcubemaptexturesize,maxdrawbuffers,ma xfragmentinputcomponents,maxfragmentuniformblocks,maxfragmentuniformcom ponents,maxfragmentuniformvectors,maxprogramtexeloffset,maxrenderbuffer size,maxsamples,maxtextureimageunits,maxtexturelodbias,maxtexturesize,m axtransformfeedbackinterleavedcomponents,maxtransformfeedbackseparateat tribs,maxtransformfeedbackseparatecomponents,maxuniformblocksize,maxuni formbufferbindings,maxvaryingcomponents,maxvaryingvectors,maxvertexattr ibs,maxvertexoutputcomponents,maxvertextextureimageunits,maxvertexunifo rmblocks,maxvertexuniformcomponents,maxvertexuniformvectors,maxviewport dims,minprogramtexeloffset,oeselementindexuint,oesstandardderivatives,o estexturefloat,oestexturefloatlinear,oestexturehalffloat,oestexturehalf floatlinear,oesvertexarrayobject,performancecaveat,redbits,renderer,sha dinglanguageversion,stencilbits,unmaskedrendererwebgl,unmaskedvendorweb gl,vendor,version,vertexshaderhighfloatprecision,vertexshaderhighfloatp recisionrangemax,vertexshaderhighfloatprecisionrangemin,vertexshaderhig

hintprecision,vertexshaderhighintprecisionrangemax,vertexshaderhighintprecisionrangemin,vertexshaderlowfloatprecision,vertexshaderlowfloatprecisionrangemax,vertexshaderlowfloatprecisionrangemin,vertexshaderlowintprecision,vertexshaderlowintprecisionrangemax,vertexshaderlowintprecisionrangemin,vertexshadermediumfloatprecision,vertexshadermediumfloatprecisionrangemax,vertexshadermediumfloatprecisionrangemin,vertexshadermediumintprecision,vertexshadermediumintprecisionrangemax,vertexshadermediumintprecisionrangemin,webgl,webglcompressedtextures3tc,webglcompressedtextures3tcsrgb,webgldebugrendererinfo,webgldebugshaders,webgldepthtexture,webgldrawbuffers,webgllosecontext,webgl2,webkitexttexturefilteranisotropic,webkitwebglcompressedtextures3tc,webkitwebgldepthtexture,webkitwebgllosecontext.

C.2 Features

The following attributes are related to the overall features of browser.

adblock,applicationcache,backgroundsize,blending,bluetooth,borderimage,borderradius,boxshadow,canvas,canvaswebp,canvasblending,canvaswinding,credentials,cssanimations,csscolumns,cssgradients,cssreflections,csstransforms,csstransforms3dc,csstransitions,draganddrop,flexbox,flexboxlegacy,fontface,generatedcontent,getbattery,getgamepads,getusermedia,hashchange,history,hsla,imghash,inlinesvg,installedfonts,installedplugins,javaenabled,js,mediadecvices,mimetypes,multiplebgs,opacity,permissions,postmessage,presentation,registerprotocolhandler,requestmediakeysystemaccess,requestmidiaccess,rgba,sendbeacon,serviceworker,shockwaveflash,smil,svg,svgclippaths,textshadow,towebp,unregisterprotocolhandler,usb,vibrate,websqldatabase,webworkers,webkitgetusermedia,webkitpersistentstorage,webkittemporarystorage,webrtc,websockets.

C.3 Media

The following attributes are related to the audio and video features of a browser.

ac-baselatency,ac-channelcount,ac-channelcountmode,ac-channelinterpretation,ac-maxchannelcount,ac-numberofinputs,ac-numberofoutputs,ac-samplerate,ac-state,an-channelcount,an-channelcountmode,an-channelinterpretation,an-fftsize,an-frequencybincount,an-maxdecibels,an-mindecibels,an-numberofinputs,an-numberofoutputs,an-smoothingtimeconstant,audioogg,avc1.42c00d,avc1.42e01e(mp4a.40.2),codecs1,dynamiccompressor,h264,hybridos

cillator,mp3,mp4a.40.2,mpeg,opus,oscillator,theora,videomp4,videoogg,vorbis(ogg),vorbis(vp8),vorbis(vp9),vorbis(wav),wav,webm,wm4a.

C.4 Input/Output

The attributes below are related to the input and output.

windowstate,outerheight,outerwidth,innerheight,innerwidth,width,height,availablewidth,availableheight,colordepth,keytimes,mouse,orientation,scrolls,maxtouchpoints,touchevent,touchstart,speakersinstalled,webcamsinstalled,microphonesinstalled.

C.5 Network

The attributes below are related to the network interface and protocols.

downlink,effectivetype,isproxied,istor,isusingtorexitnode,localip,onchange,publicipv4,publicipv6,rtt.

C.6 Miscellaneous

The attributes listed here are of various types and do not belong to any of the categories described in the thesis.

appcodename,batterylevel,charging,chargingtime,charset,collecttime,cookieenabled,cpucores,dischargingtime,donottrack,geolocation,graphicscardvendor,hardwareconcurrency,hastimezonemismatch,incognito,indexeddb,jsheapsizelimit,languages,localstorage,navigator,online,opendatabase,platform,product,productsub,referrer,renderer,sessionstorage,timestamp,timezone,totaljsheapsize,usedjsheapsize,useragent,vendor,vendorsub.

D Fingerprinting Test Code

The scripts executed by our fingerprintable website and used to perform the experiments described in Chapter 4 were gathered from the following websites.

- Most scripts were obtained from https://clientjs.org
- The script used to detect WebGL features was obtained from https://github.com/spleennooname/GLeye
- The script used to detect WebRTC features was obtained from https://github.com/muaz-khan/DetectRTC

Some scripts were modified to suit our testing. All the code we used for testing is available at our website https://fingerprintable.org. The source code of the fingerprintable test page is listed below.

```
<html>
<head>
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.1.1/jquery.
    ↪ min.js"></script>
</head>
<script type="text/javascript" src="client.js"></script>
<script type="text/javascript" src="RTC.js"></script>
<script type="text/javascript" src="GL.js"></script>
<body>
<div align="center">
<table id ="myTable" class = "fingerprint">
<link rel="stylesheet" type="text/css" href="fingerprint.css" />
<script type="text/javascript" src="fingerprint.js"></script>
</table>
</div>
</body>
</html>
```

E Browser Versions, OS Versions and Device Specifications

E.1 Browser and OS Versions

Table 2 lists operating systems and browsers that were tested in the experiments described in Chapter 4.

Table 2: Browser and OS Details

Browser	OS
Desktop	
Chrome 56.0.2924.87 (64-bit)	Windows 10.0.15063 Build 15063
Microsoft Internet Explorer 11.576.14393.0	Windows 10.0.15063 Build 15063
Firefox 51.2 (32-bit)	Windows 10.0.15063 Build 15063
Microsoft Edge 38.14393.0.0	Windows 10.0.15063 Build 15063
Safari 10.0.3 (12602.4.8)	macOS Sierra 10.12.3
Mobile	
Chrome 56.0.2924.87	Android 7.0 (Build 39.2.A.0.374)
Safari 602.1	iOS 10.2.1(14d27)
Opera Mini 22.0.2254.113472	Android 7.0 (Build 39.2.A.0.374)
Firefox 51.0.3	Android 7.0 (Build 39.2.A.0.374)
Microsoft Edge 38.14393.693.0	Windows 10 Mobile (Build: 10.0.14393.693)

E.2 Summary

Table 3 provides a summary of the specifications of the four desktop and five mobile devices used in the experiments described in Chapter 4.

Table 3: Specifications of devices used for experiments

Device	CPU	GPU
Desktop		
Asus (Windows)	Intel Core i7-4720HQ 2.6GHz	NVIDIA GeForce GTX 960M
HP (Windows)	Intel Core i5-5200U 2.2GHz	Intel HD Graphics 5500
Macbook (macOS)	Intel Core i5 2.7GHz	Intel Iris Graphics 6100
Macbook (macOS)	Intel Core i7 2.7GHz	Intel HD Graphics 530
Mobile		
Sony (Android)	Qualcomm Snapdragon 820 64-bit	Adreno 530
Samsung (Android)	Qualcomm Snapdragon 801 2.5GHz	Adreno 330
iPhone(iOS)	A8 chip 64-bit	PowerVR GX6450
iPhone (iOS)	A10 chip 64-bit	PowerVR Series7XT Plus
Microsoft (Windows)	Qualcomm Snapdragon 400 1.2GHz	Adreno 305
Microsoft (Windows)	Qualcomm Snapdragon 200 1.2GHz	Adreno 302

E.3 Details

Detailed specifications of the four desktop and five mobile devices used in the experiments described in Chapter 4 are given below.

Desktop

Windows computer 1
OS: Microsoft Windows 10.0.15063 Build 15063
System Manufacturer: ASUSTeK COMPUTER INC.
System Model: G551JW
CPU: Intel(R) Core(TM) i7-4720HQ CPU @ 2.60GHz, 2594 Mhz, 4 Core(s)
GPU: NVIDIA GeForce GTX 960M
RAM: 16.0 GB
Storage: HDD 1.0 TB

Windows computer 2
OS: Microsoft Windows 10.0.15063 Build 15063
System Manufacturer: HP
System Model: HP Pavilion Notebook
CPU: Intel(R) Core(TM) i5-5200U CPU @ 2.20GHz, 2195 Mhz, 2 Core(s)
GPU: Intel(R) HD Graphics 5500
RAM: 12.0 GB
Storage: HDD 1.0 TB

Mac computer 1
OS: macOS Sierra (version 10.12.3)
System Manufacturer: Apple Inc.
System Model: MacBookPro12,1
CPU: Intel Core i5 2.7GHz
GPU: Intel Iris Graphics 6100 1536 MB
RAM: 8.0 GB
Storage: SSD 256 GB

Mac computer 2
OS: macOS Sierra (version 10.12.3)
System Manufacturer: Apple Inc.
System Model: MacBookPro13,3
CPU: Intel Core i7 2.7GHz

GPU: Intel HD Graphics 530
RAM: 16.0 GB
Storage: SSD 256 GB

Mobile

Android phone 1
OS: Android 7.0 (build 39.2.A.0.374)
System Manufacturer: Sony Mobile Communications Inc.
System Model: Sony Xperia XZ (F8332)
CPU: Qualcomm® Snapdragon™ 820, 64-bit processor
GPU: Adreno 530
RAM: 3.0 GB
Storage: SD 64.0 GB

Android phone 2
OS: Android 6.0.1 (build 23.5.A.1.291)
System Manufacturer: Sony Mobile Communications Inc.
System Model: Sony Xperia Z3 (D6633)
CPU: Qualcomm Snapdragon 801 2.5 GHz Quad-core
GPU: Adreno 330 (component of CPU)
RAM: 3.0 GB
Storage: SD 16.0 GB

iPhone 1
OS: iOS 10.2.1 (14d27)
System Manufacturer: Apple Inc.
System Model: iPhone 6 Plus (A1524)
CPU: A8 chip with 64-bit architecture
GPU: PowerVR GX6450
RAM: 1.0 GB
Storage: SD 32.0 GB

iPhone 2
OS: iOS 10.2.1 (14d27)
System Manufacturer: Apple Inc.
System Model: iPhone 7 Plus (A1784)

CPU: A10 chip with 64-bit architecture
GPU: PowerVR Series7XT Plus
RAM: 3.0 GB
Storage: SD 128.0 GB

Windows phone 1
OS: Windows 10.0.14393.0 (32-bit)
System Manufacturer: Microsoft
System Model: Microsoft Lumia 640 XL
CPU: Qualcomm Snapdragon 400 1.2GHz
GPU: Adreno 305
RAM: 1.0 GB
Storage: SD 8.0 GB

Windows phone 2
OS: Windows 8.1 6.3.9600 (32-bit)
System Manufacturer: Microsoft
System Model: Microsoft Lumia 435
CPU: Qualcomm Snapdragon 200 1.2GHz
GPU: Adreno 302
RAM: 1.0 GB
Storage: SD 8.0 GB

F WebRTC Leaks Test Code

The script used on https://fingerprintable.org/webrtcleaks for the experiments described in Chapter 5, was obtained from https://github.com/diafygi/webrtc-ips and https://browserleaks.com/webrtc. The script was used to discover client IP address(es) through the WebRTC API and it is given below.

```
!function(){function a(a)
{function b(b)
{try{var d=/([0-9]{1,3}(\.[0-9]{1,3}){3}|[a-f0-9]{1,4}(:[a-f0-9]{1,4}){7})/.
    ↪ exec(b)[1];void 0===c[d]&&a(d),c[d]=!0}catch(a){}}
var c={},d=window.RTCPeerConnection||window.mozRTCPeerConnection||window.
    ↪ webkitRTCPeerConnection;if(!d)var g,h={optional:[{RtpDataChannels:!0}]},
    ↪ i={iceServers:[{urls:"stun:stun.l.google.com:19302"}]};try{g=new d(i,h)
    ↪ }catch(a){return void e()}g.onicecandidate=function(a){a.candidate&&b(a.
    ↪ candidate.candidate)},g.createDataChannel(""),g.createOffer(function(a)
    ↪ {g.setLocalDescription(a,function(){},function(){})},function(){},
    ↪ setTimeout(function(){var a=g.localDescription.sdp.split("\n");a.
    ↪ forEach(function(a){0===a.indexOf("a=candidate:")&&b(a)})},1e3)}
function b(a,b,c)
{return a+(c?"</a>":"")+"   "}
function d(a)
{g=a.match(/^(192\.168\.|169\.254\.|10\.|172\.(1[6-9]|2\d|3[01]))/)?"#local_ip
    ↪ ":a.match(/^[a-f0-9]{1,4}(:[a-f0-9]{1,4}){7}$/)?"#ipv6":"n/a","n/a"==$(
    ↪ g).text()&&($(g).empty(),$(g).parent().removeClass("none")),"#local_ip
    ↪ "==g?$(g).append(b(a,"_local",!1)):"#ipv6"==g&&$(g).append(b(a,"x",!1))
    ↪ }
function e()
{if(window.RTCIceGatherer)try
{var a=new RTCIceGatherer({gatherPolicy:"all",iceServers:[{urls:Array.from("
    ↪ turn:numb.viagenie.ca:3478?transport=udp").join(""),username:Array.from
    ↪ ("admin@fingerprintable.org").join(""),credential:Array.from("007").
    ↪ join("")}]}),b={};a.onlocalcandidate=function(a)
{setTimeout(function()
{if(a.candidate.ip&&"relay"!=a.candidate.type){var c
    ↪ =/([0-9]{1,3}(\.[0-9]{1,3}){3}|[a-f0-9]{1,4}(:[a-f0-9]{1,4}){7})/.exec(
    ↪ a.candidate.ip)[1];void 0===b[c]&&d(c),b[c]=!0}
},300)}}
catch(a){}
/*alert(a);*/}
var g;a(function(a){d(a)});}();
```